VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Praktiky DevOps a jejich využití při vývoji softwaru

DevOps Practices and Their Usage in Software Development

2017 Bc. Václav Vaněk

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Václav Vaněk**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Praktiky DevOps a jejich využití při vývoji softwaru
DevOps Practices and Their Usage in Software Development

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je zmapování praktik využívajících se v rámci DevOps. Analýza dostupných softwarových nástrojů pro tyto praktiky. Jejich využití ve firemním prostředí a následná analýza celkového dopadu na softwarový vývoj a samotný vyvíjený software.

Práce bude obsahovat:
1. Popis samotného DevOps a využívaných praktik.
2. Analýzu dostupných nástrojů a jejich začlenění do DevOps.
3. Analýzu a popis nasazení vypraných nástrojů do firemního vývojového procesu dle zásad DevOps.
4. Analýzu dopadu změn na vývojový proces.

Seznam doporučené odborné literatury:

[1] ERICH, Floris, Chintan AMRIT a Maya DANEVA. Cooperation between information system development and operations. In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '14 [online]. New York, New York, USA: ACM Press, 2014, s. 1-1 [cit. 2016-03-08]. DOI: 10.1145/2652524.2652598. ISBN 9781450327749. Dostupné z: http://dl.acm.org/citation.cfm?doid=2652524.2652598
[2] ERICH, Floris, Chintan AMRIT a Maya DANEVA. A Mapping Study on Cooperation between Information System Development and Operations [online]. s. 277 [cit. 2016-03-08]. DOI: 10.1007/978-3-319-13835-0_21. Dostupné z: http://link.springer.com/10.1007/978-3-319-13835-0_21
[3] HUMBLE, Jez a David FARLEY. Continuous delivery: reliable software releases through build, test, and deployment automation. Upper Saddle River, NJ: Addison-Wesley, 2010. ISBN 9780321601919.
[4] HUTTERMANN, Michael. DevOps for developers. New York: Distributed to the book trade worldwide by Springer Science Business Media New York, c2012. Expert's voice in Web development. ISBN 14-302-4569-7.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28. dubna 2017 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 28. dubna 2017

.....

VRK plus s.r.o.
Fr. Lýska 1605/3
700 30 Ostrava-Bělský Les
IČ: 60321580, DIČ: CZ60321580

Rád bych poděkoval Ing. Davidu Ježkovi, Ph.D. za vedení a odbornou pomoc při tvorbě této diplomové práce. Dále bych rád poděkoval Mgr. Robertu Kubáčkovi a celému kolektivu firmy VRK plus s.r.o. za poskytnuté prostředí pro reálné zkoušení praktik a postupů.

Abstrakt

Cílem této diplomové práce je popsat aktuální trend – DevOps přístup k vývoji SW. Kromě popisu samotného přístupu DevOps a využívaných praktik se práce dále zaměřuje na analýzu vývoje a provozu ve firmě VRK plus s.r.o.

S použitím konkrétních nástrojů jako Jenkins, Git, Maven a technik jako kontinuální integrace a doručování se v reálném prostředí malé firmy podařilo zvýšit rychlost vývoje, zjednodušit testování a zvýšit kvalitu celého vývojového procesu.

Dále je v práci popsáno použití přístupu „Infrastruktura jako kód“ a použití nástroje Puppet pro správu a konfiguraci několika serverů a nástroje Vagrant pro snadné testování konfigurace. V neposlední řadě se práce zabývá použitím praktik a nástrojů pro centrální logování. S použitím nástrojů Logstash a Graylog se výrazně zjednodušil přístup k informacím generovaným běžícími aplikacemi.

Popsané praktiky a nástroje jsou ve firmě nasazeny a jsou připraveny pro další rozvoj.

Klíčová slova: DevOps, Kontinuální integrace, Kontinuální doručování, Jenkins, Infrastruktura jako kód, správa serverů, Puppet, Centrální logování, GrayLog

Abstract

The objective of the thesis is to describe the actual trend - DevOps approach to software development. Except the decription of the DevOps approach and practices the thesis is focused on analysis of development and operations in the company VRK plus s.r.o.

The use of specific tools such as Jenkins, Git, Maven and techniques as continuous integration and delivery in small business has resulted in faster development, easier testing and improved quality of the entire development process.

It also describes the use of Infrastructure as Code approach and the use of Puppet for managing and configuring several servers and Vagrant for easy configuration testing. Last but not least, the thesis deals with the use of practices and tools for central logging. Using Logstash and Graylog tools has greatly simplified access to information generated by running applications.

The described practices and tools are deployed in the company and are ready for further development.

Key Words: DevOps, Continuous Integration, Continuous Delivery, Jenkins, Infrastructure as Code, Server administration, Puppet, Central logging, GrayLog

Obsah

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
CD	–	Continuous Delivery
CI	–	Continuous Integration
CVS	–	Concurrent Version System
Dev	–	software DEVelopers
FTP	–	File Transfer Protocol
HTTP	–	Hypertext Transfer Protocol
HW	–	HardWare
IaC	–	Infrastructure as Code
IDE	–	Integrated Development Environment
IM	–	Instant Messaging
IRC	–	Internet Relay Chat
Ops	–	information technology OPerations
OS	–	Operating System
PaaS	–	Platform as a Service
RSS	–	Rich Site Summary
SCM	–	Source Code Management
SCM	–	Source Control Management
SCP	–	Secure Copy Protocol
SCP	–	Secure Copy Protocol
SRE	–	Site Reliability Engineering
SSH	–	Secure SHell
SSL	–	Secure Sockets Layer
SW	–	SoftWare
TCP	–	Transmission Control Protocol
URL	–	Uniform Resource Locator

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Tato diplomová práce se zabývá aktuálním trendem DevOps, který je dalším krokem v agilním vývoji pro rychlejší dodání produktu, snadnější správu systému, zvýšení kvality software a zefektivnění fungování společnosti zabývající se vývojem a provozem software.

Kapitola 2 obecně popisuje pojem DevOps, jeho význam, původ a historii. Dále se zmiňuje o několika špatných pohledech na tento trend, současném dění a náhledu do budoucnosti.

Následující kapitola se zaměřuje na oblasti, na které je třeba se při praktikování DevOps soustředit. Kromě popisu těchto oblastí je v další části popsáno několik nejvýznamnějších praktik.

Kapitola 4 se zabývá nekonečným cyklem vývoje a provozu software. V této kapitole jsou jednotlivé kroky cyklu vysvětleny a je upřesněn jejich vztah k DevOps.

Dvě další kapitoly popisují reálně řešené problémy při vývoji a provozu software ve firmě VRK plus s.r.o. V první z těchto dvou kapitol je pro definování kontextu popsána firma a její produkty, kapitola dále popisuje jak analýzu, tak řešení konkrétních problémů s konkrétními nástroji a postupy. Druhá kapitola se zabývá analýzou dopadu provedených změn z předchozí kapitoly na vývojový proces. Popisuje několik reálných událostí, které se zavedenými změnami nenastanou anebo bude jejich řešení mnohem jednodušší a rychlejší. V další podkapitole jsou popsány nejvýraznější změny během této práce – počet sestavení, doba do nasazení změn a přístup k informacím z běhu aplikací. Další podkapitoly dokumentují činnosti, které už byly řešeny s použitím DevOps přístupů a nástrojů.

2 DevOps

Tato kapitola popisuje aktuální termín DevOps, jeho vznik, hlavní myšlenky a historii. Kapitola se také zabývá aktuálností trendu DevOps společně s náhledem do budoucnosti.

Výraz DevOps vznikl jako složenina dvou slov z oblasti vývoje software:

- Dev - software DEVelopers - vývoj
- Ops - information technology OPerations - provoz

Toto spojení bylo poprvé použito v roce 2009 během konference DevOpsDays (viz kapitola 2.4 Historický vývoj DevOps na straně 16). Složení výše zmíněných výrazů ilustruje i hlavní myšlenku DevOps - spojení, komunikaci a **spolupráci** dvou (a více) týmů/skupin lidí, které se zabývají vývojem a provozem software.

DevOps není žádná konkrétní specifikace nebo postup. Pod pojmem DevOps se skrývá celá řada technik, postupů, rad, nástrojů, procesů a myšlenek pro všechny zúčastněné od vývojářů přes testery a administrátory až po management a zákaznickou podporu. Hlavním cílem je poskytnout podporu pro rychlý vývoj a spolehlivý provoz. Rychlý vývoj a nasazování nových funkcí přináší získání **zpětné vazby** a tím zisk cenných podnětů při vývoji software. Zákazník ocení **spolehlivý provoz** bez výpadků a rychlou reakci na změny a získá tak větší pocit důvěry a jistoty [1].

DevOps myšlenky, tím že přesně nepředepisují žádné postupy ani nástroje, jsou otevřeny pro všechny společnosti, které se podílí na vývoji nebo provozu software. Mírnou výhodu mají firmy, které se zaměřují na online aplikace. Ale i v offline aplikacích je možné techniky DevOps úspěšně používat [2].

Podobně jako agilní vývoj i DevOps dává možnost dosáhnout různých cílů různým způsobem. Míra použití DevOps se nedá hodnotit. Mezi důvody, proč nelze dosáhnout spolehlivého hodnocení, patří neexistence přesné definice, standardu či jiné formalizace. V knize DevOps for dummies od IBM [4] stojí:

> “ *DevOps isn’t the goal. It helps you reach your goals.* ”

Výše zmíněné tvrzení dokazuje složitost a částečně i nemožnost hodnocení.

2.1 Týmy vývoje a provozu

Vývoj a provoz jsou dva hlavní týmy, které stojí za dodáním aplikace. DevOps se dotýká i jiných týmů (např. zajištění kvality, management, podpora). Ale hlavní podstatou je propast mezi týmem vývoje a provozu.

2.1.1 Tým vývoje

Hlavní složkou této skupiny jsou vývojáři, kteří vytvářejí aplikační kód. Do této skupiny ale patří i aplikační testeři, analytici, návrháři rozhraní, databázoví specialisté, specialisté na požadavky a architekti. Všichni tito lidé se snaží pracovat na svých úkolech, které většinou přímo souvisejí s vyvíjeným produktem a reflektují konkrétní požadavky zákazníka.

2.1.2 Tým provozu

Do týmu provozu se řadí všichni lidé, kteří se starají o dodávku aplikace zákazníkovi a její bezproblémový chod. Do této skupiny patří například administrátoři, síťoví správci, bezpečnostní analytici, výkonností testeři a správci databází.

Tuto skupinu spojuje to, že jejich úkolem, na rozdíl od první skupiny, není poskytnout funkcionalitu, ale zajistit, aby tato funkcionalita byla dostupná ve správný čas, správným lidem a s požadovanou kvalitou. Zajišťují provoz, instalaci a aktualizace serverů, síťové infrastruktury, bezpečnostních řešení apod.

2.2 Rozpor mezi vývojem a provozem

V tradičních společnostech fungují týmy vývoje a provozu samostatně. Důsledkem oddělení je slabá komunikace a nespolupráce. Obě skupiny mají samostatné nástroje, techniky a postupy. Na první pohled se může zdát, že vývoj má za cíl dodávat nové funkcionality (případně opravovat existující) a provoz zajišťovat stabilitu. Tyto dva cíle jsou ale v podstatě protichůdné. S novými funkcionalitami je riziko nestability vždy větší. Ale skutečný cíl je pro obě skupiny stejný - **poskytnout zákazníkovi co nejkvalitnější aplikaci** [1].

Kvůli špatně nastaveným cílům vzniká mezi vývojem a provozem úzké (někdy skoro neprůchodné) spojení, které snižuje kvalitu vyvíjené aplikace. Následkem je také to, že neexistují lidé (nebo týmy), kteří by aplikaci znali z obou pohledů. Špatný vztah může zavinit i management nebo oddělení pro zajišťování kvality nevhodným nastavením měřítek vyhodnocování a porovnávání obou oddělení.

Ke sváru obou týmů mohou přispět i špatně vytvořené procesy. Například špatně nastavený proces uvolňování nové verze. Tento proces může počítat s uvolňování nové verze vždy v pátek. To pro vývojáře znamená, že v tento den musí dodělat požadovanou funkčnost a těsně před odchodem ji odevzdat provozu k nasazení do testovacího respektive produkčního prostředí. Pro oddělení provozu to tak může znamenat večer nebo celý víkend v práci, aby dokázali novou verzi úspěšně nasadit nebo se vrátit k původní verzi a zálohám.

Techniky agilního vývoje jsou většinou dobře známé vývojářům, ale bohužel se tyto agilní myšlenky nepřenesly do celých firem [3]. Z toho pramení další nesoulad mezi těmito týmy.

Tým vývojářů je zodpovědný za analýzu, design a vytváření aplikace, chce ve shodě s agilním manifestem [12] rychle a pružně reagovat na změny, které po nich vyžaduje zákazník. Na druhé

straně tým administrátorů, zodpovědných za chod a podporu aplikace v produkčním prostředí naopak změny nevítá, protože z častých a špatně kontrolovatelných změn pramení nejistota a větší šance na pád systému, ohrožení funkčností, narušení bezpečnosti, případně i ztrátu důležitých dat.

Pokud například vznikne v systému problém, ke kterému se nedá rychle a jednoduše přiřadit zodpovědná osoba potažmo zodpovědný tým, vzniká konflikt. Případný vznik konfliktu mezi týmy má špatný vliv na celkový stav aplikace a pracovní morálku a budoucí spolupráci.

Hlavním cílem DevOps je přimět oba týmy ke spolupráci, ke sdílení nástrojů i znalostní báze. To vše ku prospěchu kvality, rychlosti a stability aplikace a následné spokojenosti zákazníka.

2.3 Nevhodný pohled na DevOps

V některých společnostech, které se pokusily přistoupit na koncepty DevOps, došlo k nepochopení hlavní myšlenky – komunikace a spolupráce.

V některých firmách existují pozice **DevOps specialista**, případně celá oddělení s tímto titulem. Paradoxně je to většinou proti hlavní zásadě DevOps. Hlavním cílem obsaženým už v názvu je komunikace a spolupráce mezi týmy. Vznikem nového týmu se dvěma týmům většinou lépe nespolupracuje - naopak může dojít k ještě většímu rozpolcení a rozdělení [1], [5].

Nicméně existují i případy, kde pozice DevOps specialisty může fungovat. V těchto pozicích lze fungovat jako evangelista, který má za úkol šířit DevOps myšlenky a podporovat týmy v komunikaci.

DevOps není ani žádnou přesnou specifikací, proto lze na tuto problematiku nahlížet pod mnoha úhly a je na každé firmě nebo týmu implementovat jednotlivé myšlenky, procesy a využít ty nástroje, které pomůžou být efektivní a rychleji plnit své úkoly.

DevOps hodně zakládá na zkušenostech týmů a jejich členů. Pokud je v týmu jen jeden člověk, který má o myšlenky DevOps zájem, tak to většinou nebude fungovat ani z části. Je třeba získat podporu v celém týmu a managementu. Správně nastavený DevOps je decentralizovaný a otevřený všem členům týmu, potažmo celé organizaci napříč různými týmy [6].

I označení nástroje jako nástroje pro DevOps může být velmi zavádějící. Opět je důležité zopakovat hlavní myšlenku – spojování lidí a komunikaci. Správný nástroj může tyto myšlenky **podpořit**, ale bez podpory lidí, procesů a fungující firemní kultury to nepůjde.

2.4 Historický vývoj DevOps

Historie DevOps začíná rokem 2008 a pokračuje do současnosti. Podle aktuálního trendu (viz obrázek 2) zájem o DevOps stále roste.

Rok 2008 V tomto roce se Patrick Debois ve své přednášce v rámci Agile 2008 v Torontu [3] poprvé veřejně zmínil o problému v komunikaci mezi odděleními pro vývoj, provoz a podporu. Hlavní myšlenka této prezentace byla o tom, že vývojáři, případně lidé zodpovědní za testování, dokážou pracovat mnohem více agilně oproti lidem zabývající se prací s infrastrukturou, řízením a obchodem. Jeho cílem tedy bylo posunout agilní principy dál za hranice vývoje.

Závěrem této prezentace bylo definování tří oblastí, ve kterých je třeba s touto problematikou začít pracovat:

1. technická – nástroje, schopnosti a možnost interakce,
2. projektová – komunikace mezi lidmi,
3. operační.

Rok 2009 Jedna z nejvýznamnějších událostí pro DevOps byla v tomto roce přednáška Johna Allspawa a Paula Hammonda „10+ Deploys per Day: Dev and Ops Cooperation at Flickr.“ na konferenci Velocity [7].

V této přednášce zazněly další klíčové myšlenky DevOps. Mimo jiné i jeden útržek z typických konverzací mezi členy vývoje a provozu.

> “ *It's not my code, it's your machines!* ”

Dále vysvětlují, že hlavním cílem provozu není udržovat provoz stabilní a rychlý. Hlavním cílem by mělo být přispívat svou prací ke spokojenosti businessu a tím i zákazníka. Toto je pro obě skupiny společné.

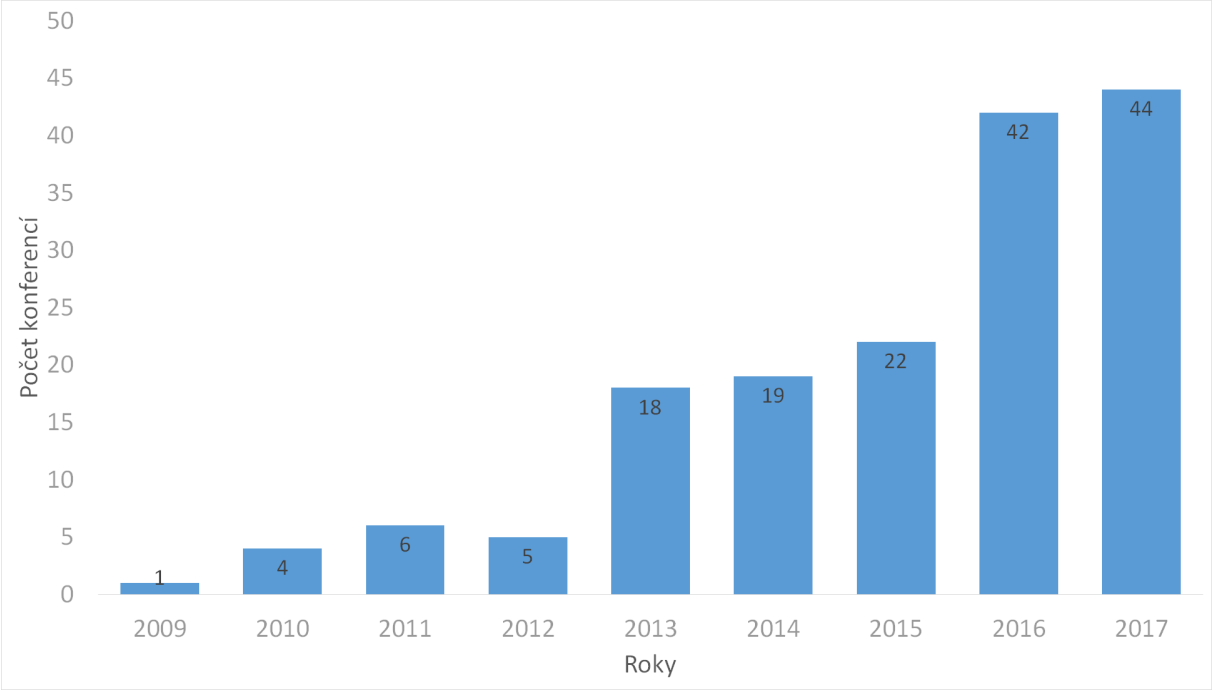
Dále naznačují i konkrétní body, kterým je třeba se začít více věnovat:

- automatizace infrastruktury,
- sdílení nástrojů pro správu verzí,
- jednokrokové sestavení a nasazení,
- přepínače vlastností,
- sdílené metriky,
- IRC a IM automaty,
- respekt,
- důvěra,
- pozitivní postoj k selhání,
- vyhnout se obviňování.

Na základě této přednášky se Patrick Debois rozhodl uspořádat konferenci, která se bude tématem DevOps zabývat – DevOpsDays.

První DevOpsDays se konal v témže roce v belgickém městě Ghent [8]. Konference s tímto názvem se konají i v současnosti a celkově už jich proběhlo více než sto a jejich počet každým rokem roste, jak ilustruje obrázek 1.

V rámci první konference zazněly mimo jiné i přednášky na téma Puppet (nástroj pro konfiguraci zdrojů), kontinuální integrace a nasazování aplikací.



Obrázek 1: Znázornění vzrůstajícího počtu konferencí DevOpsDays od počátku do současnosti

Rok 2010 až 2014 Zvětšuje se množství konaných DevOpsDays. Různé firmy přicházejí s postupy a nástroji jak praktiky DevOps podpořit. V roce 2011 vzniká open-source nástroj Vagrant a také se osamostatňuje projekt Jenkins.

Rok 2013 Společnost Puppet Labs založená v roce 2005, která vyvíjí stejnojmenný nástroj pro správu konfigurace, vydává od roku 2013 každoroční reporty o stavu DevOps. Podle prvního reportu z roku 2013 [9] bylo mezi 4000 oslovenými pracovníky z 90 zemí v oblasti softwaru 63 procent společností, které mají DevOps implementované.[1]

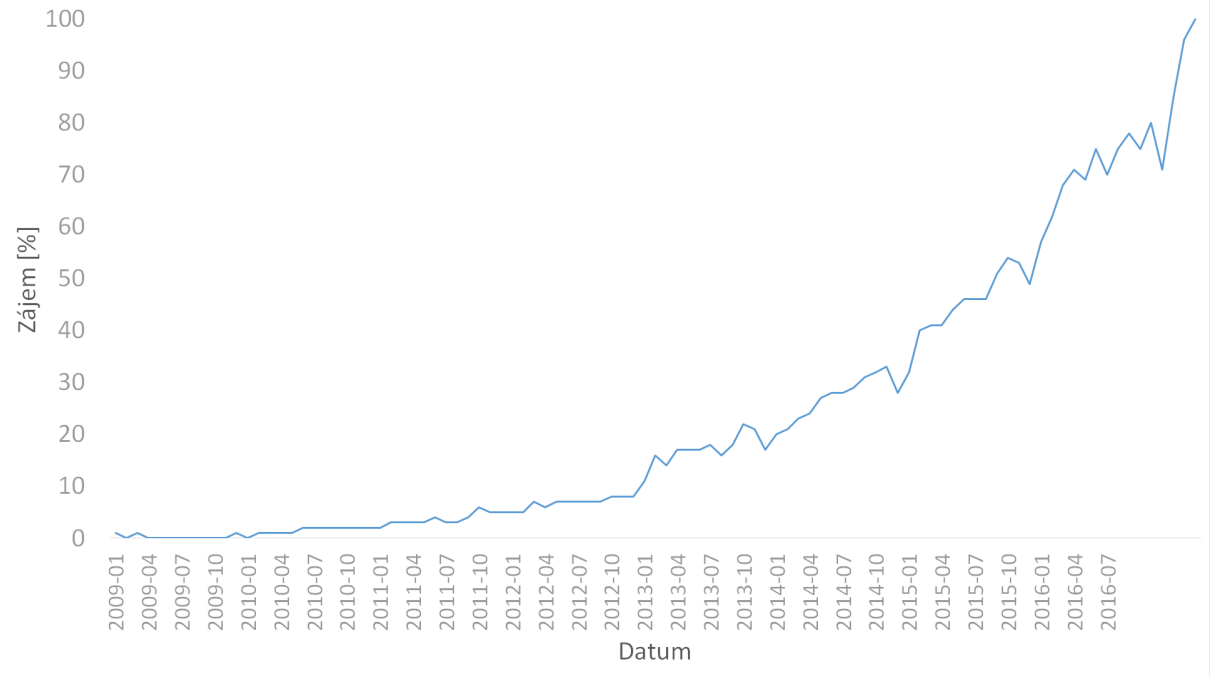
2.5 Současný stav a budoucnost

DevOps současně stále roste, myšlenky a postupy se dostávají k dalším společnostem a lidem. Web DevOps.com označil rok 2016 za rok implementace, protože DevOps se dostává do popředí zájmu [10]. Oproti minulým letům se zvýšil počet dostupných open-source nástrojů, které poskytují stejnou funkčnost jako některé placené nástroje před lety.

Velké firmy jako např. Microsoft a IBM nabízejí produkty, které pomohou k lepšímu vývoji založeném na myšlenkách DevOps. Současně se těmto tématům věnují na různých školeních a prezentacích.

[1] Tato statistika je ale do značné míry ovlivněná zvolenými respondenty a jejich počtem. Podle českého statistického úřadu bylo v roce 2013 jen v České republice přes 125 tisíc lidí zaměstnaných v IT. `https://www.czso.cz/csu/czso/ict-odbornici`

Ze statistik v Google Trends (viz obrázek 2) vyjádřujících zájem o téma DevOps v průběhu času je zřejmé, že v době psaní této diplomové práce je o toto téma relativně vysoký zájem.



Obrázek 2: Celosvětové vyhledávání klíčového slova „DevOps“ podle Google Trends[2], kde 100% znamená největší zájem v historii.

Velký posun vpřed přinášejí dostupné cloudové prostředky, se kterými se snadno a rychle pracuje. V současné době je jednoduché rychle získat velké množství výpočetních prostředků. S použitím správných nástrojů se to může dít zcela automatizovaně. Tato možnost může zajistit v krátké době velké množství výpočetního výkonu a vede i k větším možnostem pro testování (např. výkonnostního) nebo ke zvětšení konkurence (větší množství výpočetního výkonu je dostupné všem).

2.5.1 Site Reliability Engineering

Site Reliability Engineering (SRE) je jedna z možností pro rozdělení týmů. Tato myšlenka pochází z firmy Google a v dnešní době ji praktikují (alespoň částečně) i jiné velké a moderní firmy, které mají se svými on-line produkty velké množství zákazníku (Facebook, Instagram, Dropbox ale i Microsoft nebo Oracle)

Hlavní myšlenka spočívá v tom, že se tým skládá z odborníků – inženýrů, kteří jsou schopni s velkým množstvím automatizace kombinovat práci vývojářů i provozu a zajistit tak funkčnost aplikace jako celku. Na nich pak stojí také zodpovědnost za případné chyby.[3] A to do té míry, že

[2] `https://trends.google.com/trends/explore?q=%2Fm%2F0c3tq11&date=2009-01-01%202017-03-31#TIMESERIES`

[3] Někdy se podobný styl označuje také jako NoOps – toto označení využívá například Netflix [4]

dozor nad produkční aplikací si inženýři zajišťují sami a můžou rotovat ve 24/7 službě pohotovosti. Pokud je vývojář kdykoliv povinen do stanové doby reagovat na vzniklou chybu, extrémně se zvýší jeho snaha o bezproblémový kód [11].

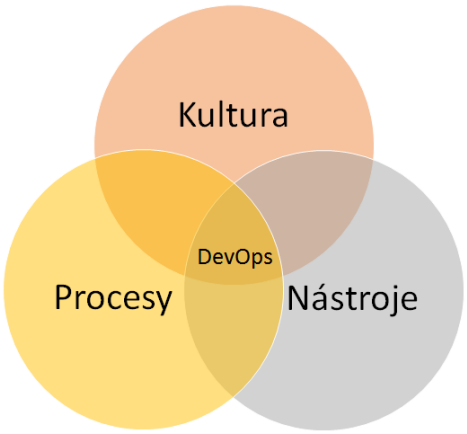
Tato cesta vývoje nicméně klade obrovské nároky na lidi, jejich znalosti a nástroje, ale je poměrně velkou zárukou kvalitně fungující aplikace. Je nutná i značná podpora infrastruktury a cloudových řešení. Nutná je i podpora ze strany managementu. SRE se samozřejmě nehodí na všechny produkty a do všech prostředí a je třeba konkrétní použití vždy individuálně zvážit.

Lze dohledat informace o tom, že s použitím SRE lze nahradit DevOps. Nicméně samotný Google to vyvrací [11] a vysvětluje, že SRE je nástavba, pokračování DevOps a že vzhledem k tomu, že od roku 2008 je DevOps pořád ve stavu nasazování, tak než všichni budou schopni překročit k SRE, bude to trvat ještě dlouho dobu.

3 Využívané praktiky a postupy

DevOps, jak už bylo zmíněno výše, nelze popsat konkrétní specifikací, ale existuje velké množství postupů, nástrojů a pravidel. Pochopení a dodržení těchto pravidel vede k lepší spolupráci a tím kvalitnějšímu produktu a spokojenému zákazníkovi. Následující kapitola se zaměří na několik konkrétních pravidel a postupů.

Rozdělení pravidel není striktní a některé činnosti přesahují do více oblastí, což je i důkaz hlavního poselství DevOps - spolupráce. Literatura (např. [4]) uvádí rozdělení všech praktik do tří oblastí (viz obrázek 3). Nicméně například v článku „What is DevOps“ [1] je zmíněno, že nejde jen o tyto oblasti, ale o komplexní propojení všech oblastí vývoje a provozu.



Obrázek 3: Ilustrace spojení tří oblastí vyskytujících se během vývoje a provozu software: kultury, procesů a nástrojů

3.1 Firemní kultura, lidé

Firemní kultura je nejdůležitější oblastí pro zlepšení vývoje a pro DevOps je zcela zásadní. Stojí u počátku samotného vzniku DevOps – problémy v komunikaci mezi oddělením vývoje a provozu.

Tato část se ale současně považuje za nejsložitější část při zavádění DevOps. Bez správné firemní kultury nemůžou nástroje ani procesy samostatně fungovat. Současně je to část, která má přesah do sociálního inženýrství, týká se lidské povahy, kterou může být velmi náročné ovlivnit nebo změnit.

Management a vedoucí na všech místech musí znát a podporovat myšlenky DevOps, musí se snažit odstraňovat bariéry mezi jednotlivými týmy. Management by neměl vytvářet hodnocení výkonu, které by bylo založeno na metrikách vedoucích ke konfliktu mezi odděleními. Například hodnocení vývojářů podle počtu funkčností, protože více rychle vyvinutých funkčností v jednom balíku změn může znamenat větší šanci na chybu. Z druhé strany může být nevhodné hodnocení

pro provozní oddělení například podle doby běhu aplikace bez poruchy. Tým provozu se může, ve snaze o co nejméně změn, držet malého počtu nasazení za určité období. Spojením těchto protichůdně nastavených hodnocení může vzniknout rozpor. Vývojáři vyvíjí překotně nové funkčnosti, ale provoz je nechce akceptovat [4].

DevOps klade velký důraz na otevřenost a důvěru mezi týmy a lidmi. Pokud si lidé důvěřují, dokážou společně dosáhnout kvalitnější výsledků, než kdyby se navzájem jen kontrolovali a hledali vzájemné nedostatky. S otevřeností a důvěrou přichází také respekt. Je důležité, aby vývojáři respektovali práci provozu a naopak. Aby se obě strany navzájem neshazovaly a byly si vědomy, že bez ani jedné strany by business nebyl schopen dostát cílům a uspokojit zákazníka.

S důvěrou ale přichází také zodpovědnost za určité činnosti nebo procesy. Zodpovědnost znamená jednak větší nároky na lidi, ale také přináší motivaci k lepším výsledkům. Předávání odpovědnosti za celou aplikaci od vývoje přes provoz jednomu týmu je praktika masivně využívaná v SRE (viz kapitola 2.5.1 Site Reliability Engineering).

Důležité také je, aby všichni účastníci vývoje a provozu věděli a chápali význam jednotlivých funkcí, ideálně i celého systému. Prvoplánově nejde o pochopení implementací všech částí systému, ale všichni by měli vědět, komu systém slouží a jak ho používá.

V roce 2013 bylo podle Puppet reportu [9] největší překážkou (48 %) pro zavádění myšlenek DevOps právě nepochopení myšlenek a neochota změnit zvyklosti.

3.2 Procesy a postupy

Procesy a postupy jsou další oblastí, do které DevOps zasahuje. Procesy nemusí být formalizované a rozhodně nesmí sloužit jako nástroj byrokracie. Nicméně existence konkrétních procesů týkajících se například řízení uvolnění nové verze aplikace, řízení změn nebo postupu opravy chyb je pro DevOps důležitá. Všichni účastníci vývoje produktu by měli být s procesy seznámeni a měli by chápat význam jednotlivých procesů.

Špatně nastavené procesy, které komunikaci napříč skupinami zhoršují (nebo přímo znemožňují), můžou vést ke špatným rozhodnutím a zbytečným ztrátám. Například pokud se vývojáři a provoz nebudou mít možnost potkávat na poradách a řešit problémy společně. Toto znemožnění komunikace může vést k tomu, že požadavek na funkci v systému, který by se dal elegantně a rychle řešit pomocí běžně používaného nástroje nebo techniky v provozu, budou vývojáři složitě a draze řešit vlastní cestou. Naopak úkol, který bude přiřazen provozu, můžou vývojáři kladně ovlivnit a dopomoct kvalitnějšímu a rychlejšímu řešení.

Dalším problémem může být například špatný proces, který přiřazuje odpovědnost osobám, kteří danou věc už nemůžou ovlivnit. Přitom by danou věc mohl zvládnout někdo jiný s menšími náklady a kvalitnějším výsledkem.

Information Technology Infrastructure Library a DevOps

Information Technology Infrastructure Library (ITIL) je souhrn doporučení a postupů jak dosahovat kvalitnějšího provozu IT. Na první pohled se může zdát, že je s DevOps nekompatibilní, protože je silně orientován na procesy zaměřené pouze na provoz systému a nejde tak v první řadě o komunikaci a propojení týmů. Na druhou stranu je ITIL na rozdíl od DevOps jasně popsaný a dokumentovaný. Při implementaci ITIL často vznikají oddělené skupiny, které zajišťují určitou činnost nebo proces. Pokud se ale zapojí myšlenky DevOps o komunikaci a sdílení informací a použití nástrojů, může dojít k harmonickému spojení. Ve výsledku tak může ITIL fungovat jako vzor procesů potřebných pro provoz software, které budou dobře fungovat i v DevOps [20].

3.3 Nástroje a technologie

Nástroje a technologie jsou neméně důležitá oblast. Nesmí se ale zapomínat, že DevOps je silně agilní a v souladu s agilním manifestem se „upřednostňují jednotlivci a interakce před procesy a nástroji“ [12]. Nástroje a technologie jsou v DevOps velkým pomocníkem, nesmí se však používat špatně nebo v místech, kde je lidská práce smysluplná a dosahuje kvalitnějších výsledků.

Nicméně některé praktiky by bez použití patřičných nástrojů, byly velmi složitě proveditelné [13]. Použité nástroje by neměly svazovat a měly by být snadno použitelné pro všechny účastníky vývoje. Skupiny by se měly navzájem o svých nástrojích informovat a nastavit tak společně fungující celek.

Výběr správných nástrojů a technologií je ale velmi složitý a staví na zkušenostech zúčastněných. Vždy je lepší používat nástroje, které daní lidé znají, než nutit nástroje nové, které mohou mít nějaké přidané vlastnosti, ale ve výsledku nic navíc nepřinesou. Pokud klíčové funkce splňuje i ten původní nástroj je ve většině případů lepší zůstat u něj. Problém může být například i tlak managementu na používání určitého nástroje, který vývojářům z objektivních důvodů nevyhovuje a nevidí v něm přínos. Nesprávné nastavení nebo použití nástroje může zpomalovat vývoj, brzdit vývojáře v jejich práci nebo také snižovat kvalitu kódu (příliš přísné nastavení kontroly kvality kódu, kde vývojáři nechápou význam jednotlivých kontrol a nepřikládají jim důležitost).

Výše zmíněné podkapitoly popisují tři hlavní oblasti DevOps: firemní kulturu a lidi; procesy a postupy; nástroje a technologie. Další část této kapitoly se zaměřuje na několik konkrétních postupů a praktik, kterou jsou pro DevOps zcela klíčové a zásadní.

3.4 Kontinuální integrace

Kontinuální integrace (Continuous Integration - CI) se jako pojem poprvé objevuje už v roce 1991 v knize Object-Oriented Analysis and Design with Applications [14]. Ve velké míře se však začala tato praktika uplatňovat až s příchodem agilního vývoje.

Hlavním cílem této praktiky je co **nejčastější spojování změn** od jednotlivých vývojářů, týmů nebo skupin do jednoho stále kompilovatelného a ideálně i funkčního celku. Za slovem nejčastější se obvykle skrývá několikrát za den, respektive po každé změně.

Čím dříve dojde ke spojení všech změn, tím je větší šance, že nevzniknou problémy, které by vznikly při dlouhém odkládání spojení. Současně to přináší obrovskou výhodu v podobě rychlé zpětné vazby v případě problémů. V případě, že problém nastane, bude naprosto jasné, po které změně k problému došlo a bude třeba vynaložit mnohem menší úsilí k identifikaci i opravě problému. Pokud by změna nebyla dlouho integrována, může na ní být založena celá řada dalších změn a tím potřebné úsilí k opravě může růst [15].

V praxi se pod pojmem kontinuální integrace skrývá na první pohled jednoduchý postup. Vývojář začne zpracovávat nový úkol. Získá verzi zdrojových kódů na své zařízení (lokální kopie), ve které jsou integrovány všechny aktuální změny jeho kolegů. Provede změny, které jsou pro splnění úkolu nutné. Jakmile svou práci dokončí, zkontroluje jestli je aplikace funkční (provede sestavení, spuštění a otestování). Po úspěšném splnění předchozího bodu svou práci synchronizuje, tak aby byla dostupná všem ostatním.

Nyní přichází čas pro automatizace. Zdrojové kódy a vše ostatní k sestavení aplikace je nyní v centrálním místě dostupné všem. Nad verzí v centrálním úložišti se spustí sestavení a testování. V případě úspěchu je změna korektně zintegrována. V případě neúspěchu dostává vývojář, který změnu vytvořil, upozornění a měl by chybu co nejrychleji odstranit [15].

Kontinuální integrace nicméně začíná ještě dříve než při samotné editaci zdrojových kódu. Na to, že všechny změny by měly být okamžitě integrovány se zbytkem, je nutné myslet už při vytváření jednotlivých úkolů a plánování. Úkoly by měly být splnitelné v krátkém časovém okamžiku (ideálně v rámci hodin), tak aby se při jejich plnění neztratil kontakt se zbytkem kódu.

3.5 Kontinuální doručování

Kontinuální doručování (Continuous delivery - CD) je další technika, která pomáhá zkrátit dobu dodání nové verze produktu a získat tak mnohem rychlejší zpětnou vazbu. Dá se říct, že je to další úroveň po CI. Pokud je možné při praktikování CI každou změnu sestavit a otestovat, tak v CD jde o možnost každou změnu snadno odeslat do různých prostředí a následně až ke klientovi.

Pro dosažení je potřeba intenzivní spolupráce všech zúčastněných a podpora automatických nástrojů. V rámci CD by měla být k dispozici možnost, kterou lze kterékoliv sestavení doručit do libovolného prostředí, a to bez potřeby hlubších znalostí toho, jak je to provedeno („one button delivery“). Naopak, pokud je na provedení akce doručení aplikace potřebný speciální tým, specialista nebo složitý a manuální postup, znamená to velký nesoulad s CD a ve většině případů i snížení četnosti a zpomalení celého procesu nasazení [16].

Dosažení CD povede ke snížení rizika, které při nasazení vzniká. Tlak na časté nasazení bude eliminovat všechny manuální kroky a tím zvyšovat míru automatizace. S malými a čas-

tými změnami také přichází snížení rizika chybovosti – menší balík změn je na chyby mnohem méně náchylný, protože změna je konkrétní a dobře pochopitelná. Pokud se provádí kontrola kódu, tak menší změny se kontrolují snadněji a zvětšuje se šance na odhalení chyby ještě před uvolněním.

Další výhodou dobře zpracovaného kontinuálního doručování je možnost se snadno, rychle a v podstatě bez rizika vrátit k předchozí verzi. Vzhledem k tomu, že rozdíl mezi verzí s chybou a poslední funkční verzí je díky malým změnám pouze v oné jedné, konkrétní a vyhledatelné chybové funkčnosti, bude návrat na nižší verzi znamenat z pohledu aplikace jen odstranění chybové vlastnosti nebo funkčnosti.

Na druhou stranu kontinuální doručování sebou přináší i rizika a zvýšené nároky. Jedním ze základních předpokladů je existence kvalitních, automaticky spustitelných a vyhodnotitelných testů. Dalším požadavkem k úspěšném fungování je podpora ze strany zákazníka. Pokud zákazník z jakýchkoliv důvodů (např. pokud se jedná o kritický software) nechce často nasazovat novou verzi tak logicky nelze zcela automatického doručování dosáhnout.

V souvislosti s tématem kontinuálního doručování se objevují dva termíny, které někteří považují za totožné, ale jiní v nich vidí podstatné rozdíly.

Vztah continuous delivery a continuous deployment

Vztah mezi „Continuous delivery“ a „Continuous deployment“ je otázkou, na kterou se těžko hledá jednoznačná odpověď. Martin Fowler [16] ale říká, že podstatný rozdíl je o tom, zda se daná verze do produkčního prostředí dostane automaticky nebo s manuálním ověřením.

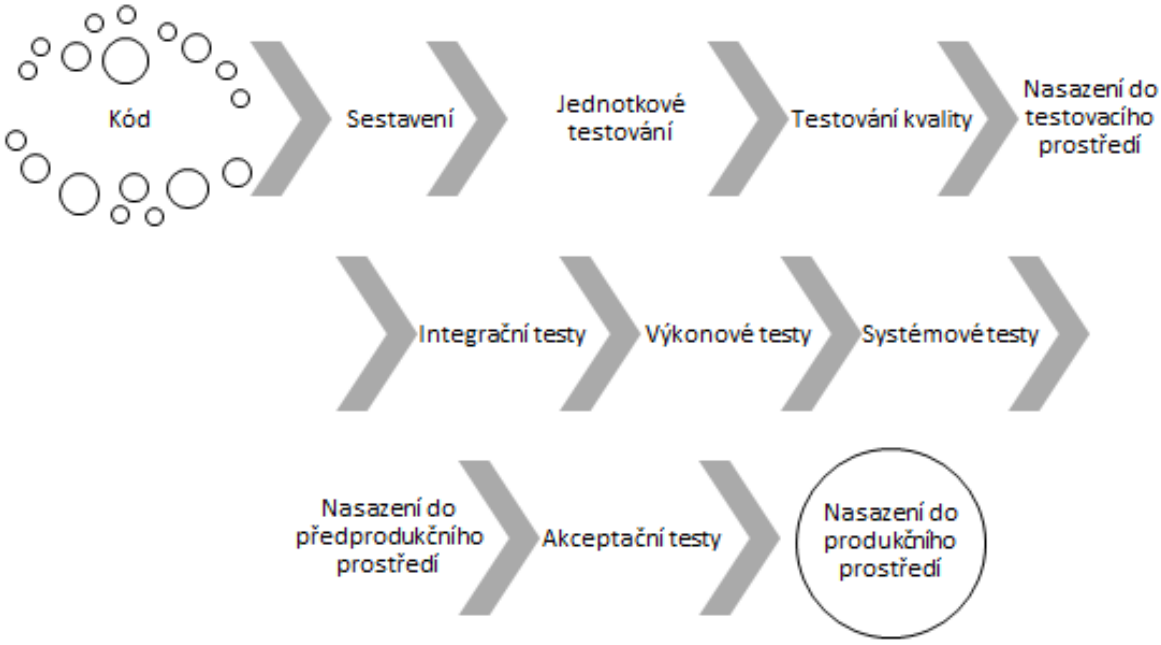
Continuous delivery – každá změna má možnost se přes různá prostředí dostat až do produkčního, ale před nasazením do produkce se čeká na manuální potvrzení.

Continuous deployment – každá změna se automaticky přes různá prostředí dostane až do prostředí produkčního.

Postup, kterým daná verze sestavení prochází, se označuje jako „deployment pipeline“ – viz obrázek 4. Ze znázornění cesty konkrétního sestavení v deployment pipeline, by mělo být vždy na první pohled zřejmé, ve které části (stage) se změna nachází. Pokud dojde k chybě, je jasné, kde k chybě došlo, a současně je dostupné i konkrétní chybové hlášení, podle kterého se dá s chybou dále pracovat. Současně chyba způsobí ukončení propagace dané verze do dalších prostředí. Takže v případě, že daná verze neprojde automatickými integračními testy, tak už nebude zbytečně zatěžovat hardware výkonnostním testováním.

Používání přepínačů funkcí

Používání přepínačů funkcí (feature toggles/flags) je jedna z praktik, která umožňuje pracovat na úkolech, které jsou složitější a vyžadují tak větší množství práce, která není proveditelná



Obrázek 4: Znázornění postupu konkrétní změny ve zdrojovém kódu aplikace přes různé stádia a prostředí

v rámci jednoho úkolů. Pokud by funkčnost nebyla dokončena, neměla by se dostat do zbytku zdrojových kódu, ale tím se porušují pravidla pro kontinuální integraci.

Aby bylo možné funkčnost, která ještě není připravená pro použíti klientem, integrovat a nasadit do produkčního prostředí, využívá se tato technika. V podstatě se jedná o možnost nastavit, které funkce se uživatelům ukážou a které zůstanou skryté. Pokud uživatel nemá k funkcionalitě přístup, může zůstat částečně nedokončena, ale kód bude integrován. V podstatě se jedná o náhradu větví z verzovacích systémů, tak aby kód mohl být neustále integrován. Této vlastnosti lze využít i pro testování nové funkcionality na menší množině uživatelů nebo jednom serveru.

3.6 Automatizace infrastruktury, infrastruktura jako kód

Po dvou praktikách, které se týkaly spíše vývoje software, popisuje tato kapitola techniku používanou v provozu.

Infrastruktura jako kód (Infrastructure as Code - IaC) je přístup, který umožňuje pracovat s infrastrukturou jako se zdrojovým kódem. Infrastrukturou je v tomto případě myšleno veškeré hardwarové i softwarové vybavení, které je potřeba pro chod a vývoj aplikace. Konkrétním příkladem může být instalace operačního systému, middleware, databázového serveru, nastavení uživatelských práv nebo umístění konfiguračních souborů [13].

Práce s infrastrukturou jako s kódem přináší spoustu výhod. **Možnost verzování** – stejně jako zdrojový kód aplikace i kód pro definice infrastruktury lze verzovat. To sebou nese možnosti jako zpětné dohledání konkrétní změny nebo schopnost se snadno vrátit k předchozí verzi infrastruktury.

Dále se infrastruktura jako kód dá snadněji a lépe testovat. Obvyklou praxí je vytvářet virtuální prostředí, do kterého se aplikuje infrastruktura a nad kterou se dají spustit automatizované testy. Tímto postupem se zaručí, že je veškerá konfigurace v kódu a nedochází k manuálním ad-hoc úpravám.

Infrastruktura popsaná kódem, který se vykonává, je **samo-dokumentující**. Při klasickém pojetí infrastruktury by se měla s každou fyzickou změnou upravit dokumentace, která danou infrastrukturu popisuje. Pokud se důrazně dodržuje využití IaC, tak jinou cestou než úpravou kódu ke změně infrastruktury dojít ani nemůže. Konkrétní příkazy lze okomentovat přímo v kódu, který tak může sloužit jako kvalitní základ pro generovanou dokumentaci.

IaC také přináší možnost použití stejné konfigurace na více místech bez vynaložení většího úsilí. Nástroje, které pro IaC existují, počítají s konfigurací velkého množství infrastruktury a serverů. Například lze servery, které mají sloužit jako databázové s konkrétní verzí databáze a nastavením, označit jako skupinu databázových serverů a nástroj pak provede jejich nastavení na všech stejně a automatizovaně.

3.7 Měření a metriky

Měření a vyhodnocovaní metrik jsou podstatnou informací, která dokáže posunout vývoj a provoz vpřed. Bez měření a vyhodnocování je totiž optimalizace v zásadě nemožná.

Problém, který nastává při tradičním použíti metrik, může být to, že metriky agregují složité věci do jednoho čísla tak, aby se dobře prezentovaly managementu nebo zákazníkům. Metriky se pak můžou stát nástrojem, se kterým se dá manipulovat tak, aby poskytoval ukazatele, které management uspokojí, i když skutečný stav projektu může být jiný.

Metriky by ale měly primárně sloužit pro zjišťování stavu aplikace, její stability, kvality, frekvence použití daných funkčností aj. Metriky by měly být **snadno dostupné všem**, i když se na první pohled může zdát zbytečné zpřístupnit například počet přihlášených uživatelů vývojářům. Zpřístupněním takové metriky získá vývojář možnost sledovat, kolik skutečných uživatelů danou aplikaci používá, a stoupne tak jeho motivace dělat svou práci kvalitně.

Například metrika počet nahlášených a vyřešených chyb může vyústit k neustálému dohadování, co je skutečně chyba a co je pouze neimplementovaná funkčnost. Metrika, která hodnotí provozní oddělení podle času bez poruchy, může znamenat opakované přehazování problému mezi provozem a vývojem nebo snížení frekvence dodávky nové verze software.

Dalším problémem může být použití metrik pro porovnání výkonnosti týmů. Pokud budou metriky postavené protichůdně nebo soupeřivě, týmy spolu přestanou komunikovat a mohou se vzájemně poškozovat.

Metriky by měly sloužit jako přínos týmu nebo přínos pro zákazníka. Například pokud je nějaká funkcionalita nadefinovaná nebo i implementovaná, ale zákazník ji nemůže využívat (není nasazená), tak nepřináší žádnou hodnotu. Proto například metrika založená na počtu zapsaných požadavků od zákazníka nepřináší zákazníkovi žádnou hodnotu [13].

DevOps se dále zaměřuje na kontinuální získávání monitorovacích dat, které jsou dostupné všem lidem, kteří se na vývoji podílejí. S používáním CI/CD je správná informace o aktuální stavu aplikace, včetně hardwarových informací, zcela nezbytná.

3.8 Nepřetržité a centrální monitorování

Nepřetržitý a dostupný přísun dat produkovaných běžícím systémem je další oblast, kterou se DevOps zabývá. Hlavní myšlenka je o tom, získat co největší množství informací, které jsou dále využitelné pro zvýšení kvality dané aplikace [20].

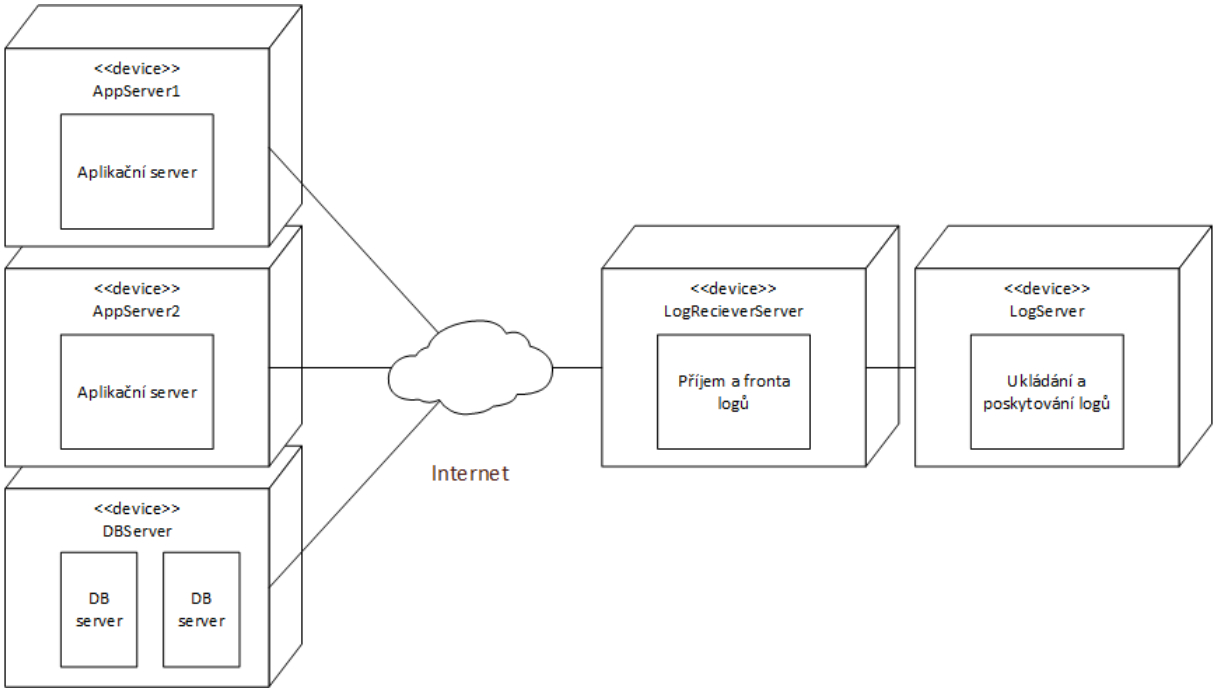
Centrální monitoring je technika, kterou se ve většině případů řeší získávání a práce s daty produkovanými za běhu systémem, aplikacemi a jejich moduly. Hlavní myšlenkou je všechny informace proudící z různých systémů produkovat v dále zpracovatelné formě, transformovat do jednotné podoby a v jednotné podobě je prezentovat v jednoduchém a snadno dostupném rozhraní (nejčastěji webová aplikace).

Použitím této techniky lze dosáhnout některých funkčností, které by bez ní nebylo možné získat. Snadněji lze řídit přístup konkrétních osob/skupin ke konkrétním informacím. Lze dosáhnout spolehlivé archivace a záloh monitorovacích dat. Tím, že všechna data budou mít stejný formát, lze využít parametrizované vyhledávání napříč velkým množstvím dat. Vyhledaná data lze třídit, lze snadno získat časové souvislosti mezi logy z různých systémů. Takto centralizovaná data lze snadněji využít pro vytváření grafů a dashboardů, které můžou poskytnout rychlý přehled o aktuální situaci.

Obecný průběh získávání monitorovacích informací je vždy stejný:

1. produkce dat v běžícím systému,
2. předzpracování na vzdáleném serveru,
3. zabezpečený přenos na centrální server,
4. zařazení do fronty,
5. záloha původních nezpracovaných dat,
6. zpracování dat (převod formátů, dopočítání, případně rozdělení),
7. využití zpracovaných dat – získání informací,
8. archivace dat,
9. po stanovené době odstranění archivovaných dat.

Je důležité nejen data získávat, ale také je systematicky uspořádat a postarat se o co nejlepší výstup přímo od zdroje, aby nebylo nutné provádět složité změny formátu v systému pro centrální



Obrázek 5: Obecné schéma centrálního logování, přenos informací z běžících serverů do centrální fronty, zpracování a následné poskytování v ucelené podobě

správu. Dále je podstatné vyřešit redundance a nesoulady v nastavení (např. lokalizace data a času, oddělení nových řádků, pořadí, ...)

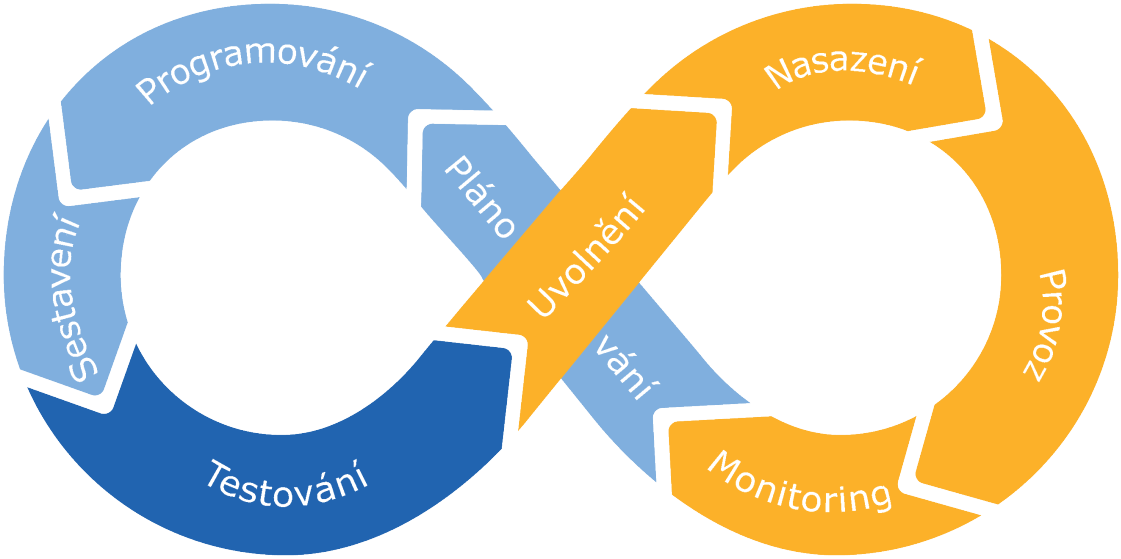
Pro správnou funkci centrálního monitorování není nutné implementovat všechny body. Vždy záleží na množství a kritičnosti přenášených dat. Nicméně je během implementace tohoto řešení třeba mít v paměti možnost budoucího rozšíření a škálování celého řešení.

4 DevOps cyklus

Tato kapitola se zaměřuje na nekonečný cyklus vývoje a provozu software. Popisuje základní fáze, ve kterých se daná verze softwaru může nacházet.

Softwarové úpravy (nové vlastnosti, opravy chyb, atd.) během svého života prochází několika fázemi, od zadání a vlastní kódování, přes sestavení, testovaní, po nasazení, samotný provoz a monitoring. V rámci DevOps lze každou z těchto částí podpořit nějakým nástrojem a postupem, který přinese zlepšení vývoje a kvalitnější aplikaci. Všechny nástroje použité v procesu by měly mít možnost integrace s jinými nástroji tak, aby došlo k jejich propojení a harmonickému fungování.

Pokud se mluví o DevOps, často se vývoj a provoz softwaru zobrazuje na smyčce – viz obrázek 6. Toto znázornění zvýrazňuje to, že činnosti na sebe navazují, a že se jedná o nekončící proces, kde je pro úspěšné propojení třeba integrace nástrojů a spolupráce obou týmu. Propojení všech fází přes obě skupiny je důležité i proto, aby nenastala situace, kdy místo smyčky vzniknou dva nespojené kruhy (vývoj a provoz), které se točí každý jinou rychlostí.



Obrázek 6: Vývoj a provoz v nekončící a stále se opakující smyčce jednotlivých fází, převzato z [17]

Následující text se zabývá jednotlivými částmi cyklu. Popisuje každou fázi jako takovou, její hlavní cíle a problémy.

4.1 Plánování

První částí, do které praktiky DevOps zasahují, je už samotné plánování úkolů, které se v rámci softwarového projektu vykonávají. Základní myšlenkou je to, aby všechny úkoly byly zapsány v jednom systému, držely jeden formát a styl. Systémů pro zpracování úkolů je celá řada, od jednoduchých, které se omezují na prosté zadání názvu a popisu, přes složitější, které podporují životní cyklus úkolu, až po ty sofistikované s velkými možnostmi plánování a přípravy. Výběr systému ovlivňuje i způsob práce a typ vyvíjeného software.

Je důležité, aby byl systém všemi lidmi zapojenými do vývoje software akceptován. Pokud někteří členové budou mít se systémem problém, je to první známka, že je něco v cyklu špatně, a může ho to celý rozbít. Systém pro úkoly nesmí být nastaven tak, aby lidi, kteří s ním pracují, zdržoval v jejich hlavní práci. Systém má sloužit jako podpora řešení úkolů – hlavním cílem musí být stále vyřešit úkoly.

Pokud jde o praktiky DevOps, je důležité, aby bylo možné systém integrovat do jiných systémů. Ideálním stavem je, pokud to systém umožní už ve své základní verzi. Použitím rozšíření výrobce nebo třetích stran s sebou přináší mírné riziko nefunkčnosti v případě aktualizací.

Kromě samotného systému pro zadávání úkolů je důležitá snaha vytvářet atomické úkoly, které jsou splnitelné v rámci krátké časové jednotky. Úkoly by měly být dostatečně popsané a doplněné o informace, které jsou při řešení úkolu žádoucí.

Následující seznam popisuje vybrané vlastnosti z oblasti systémů pro plánování, které jsou pro snadné nasazení v rámci DevOps cyklu klíčové.

Identifikátor – každý úkol musí mít jednoznačně přiřazený identifikátor, který se nemění a je pevně svázán s daným úkolem. Identifikátor musí být unikátní napříč celým systémem (v rámci různých projektů). Identifikátor pak slouží jak lidem, tak i dalším systémům, které můžou konkrétní činnost spojit s daným úkolem.

Stavy – systém pro správu úkolů musí podporovat možnost přechodu do různých stavů. Tyto stavy mají za úkol zvýšit přehled v jednotlivých úkolech, aby každý okamžitě věděl, co se má s daným úkolem dále dít.

Persistence – úkoly, které se jednou vytvořily, by měly být vyhledatelné i přes odmítnutí, či případně zrušení.

Rozšiřitelnost – systém by měl mít možnost instalace doplňků a rozšíření. Tato vlastnost je důležitá pro propojení s dalšími nástroji a úpravy systému pro použití, které vyhovuje všem zúčastněným.

Provoz by měl používat pro zadávání úkolů stejný systém jako vývoj. Samozřejmě budou mít jiné typy úloh, jiné kategorie, ale systém by měl být stejný. V případě potřeby tak nebude problém přesunout úkol z vývoje na provoz a opačně. Stejný systém také podpoří kolektivní inteligenci a vzájemné učení a respektování cizí práce – obě skupiny můžou navzájem vidět úkoly těch druhých. Tím se také podpoří motivace pro větší výkon jednotlivých lidí.

4.2 Programování

V této části cyklu vzniká programový kód. Pro většinu vývojářů je to jedna ze stěžejních částí celého cyklu vývoje software a celé své soustředění věnují právě této části. Vytvářením kódu se začínají naplňovat jednotlivé úkoly zadané ve fázi plánování.

Výběr jazyků a technologií pro tvorbu software velmi ovlivňuje další fáze DevOps. Například při volbě nekompilovaného jazyka může vypadat, že fáze sestavení nemá smysl, protože kód lze rovnou spustit. V dnešním světě programovacích jazyků už to ale není tak docela pravda. I většina nekompilovaných jazyků umožňuje a někdy i vyžaduje používat systém pro správu závislostí, rozšíření a různých preprocesorů, díky kterým je výsledný kód snadněji udržitelný a rozšiřitelný.

4.2.1 Vývojové prostředí

Programový kód vzniká ve vývojových prostředích (IDE[4]), které kombinují několik nástrojů dohromady a vývojářům ulehčují psání kódu a opakované nebo manuální činnosti. Většina běžných programovacích jazyků a platforem je podporována v několika různých vývojových prostředí.

Ideální je, pokud je celý produkt vyvíjený v jednom vývojovém prostředí a vývojáři můžou sdílet konfiguraci a nastavení. Důležité ale je, aby byl produkt na vývojovém prostředí co nejméně závislý a fungoval i bez něj. Této vlastnosti se využije v dalších fázích DevOps cyklu.

Pokud je produkt kompilovatelný jen v jednom konkrétním IDE, může to výrazně brzdit další fáze cyklu, které se kvůli tomu mohou stát velmi složité. Některé IDE vývojáře úplně odstiňují od věcí jako je kompilace, sestavení, nasazení a spuštění aplikace. Odstínění může mít za následek snížení důležitosti těchto činností v očích vývojáře a vrhat špatné světlo na lidi, kteří se těmito činnostmi zabývají v provozu.

4.2.2 Správa verzí

Software zpravidla tvoří více vývojářů a je potřeba zajistit, aby spolu mohli kód jednoduše sdílet a historizovat. Proto existují nástroje pro správu verzí (SCM[5]). Ty shromažďují všechny zdrojové kódy aplikace, umožňují historizaci a přinášejí možnost zdrojové kódy jednoduše sdílet a spolupracovat na nich. Jednotlivé přírůstky se změnami a novými zdrojovými kódy se zpravidla označují jako commity[6].

Tyto nástroje dávají povětšinou vývojářům volnost v tom, jak je budou přesně používat. Ale pro DevOps a správný vývoj je potřeba dodržovat určitá pravidla, které jsou popsána v knize [18] a [19].

Atomičnost – každý commit by měl řešit pouze jednu funkčnost, jednu změnu v systému, jeden úkol z fáze plánování. Pokud vývojář upraví více funkčností v jednu chvíli, měl

[4]IDE – Integrated Development Environment
[5]SCM – Source Code Management
[6]commit - potvrzení a uložení změn

by vytvořit více samostatných commitů. Dodržováním tohoto pravidla se usnadní práce v dalších fázích.

Izolovanost – commity by na sobě neměly být závislé, každý commit by měl být aplikovatelný samostatně. Toto pravidlo umožní nepřijmout funkčnosti, které nesplňují zadání nebo se v nich vyskytuje problém.

Popis, zpráva – ve většině verzovacích systémů je možné před odesláním commitu vytvořit zprávu, která se stane součástí daného commitu. Zprávy by měly dodržovat určitý formát a být popisné, tak aby v případě problému a zpětného dohledání bylo na první pohled jasné co, jak a kdo konkrétním commitem řešil.

Generované soubory – nemají být součástí repozitáře. Většina systémů pro správu verzí umožňuje nastavit ignorování souborů. Pokud se budou generované soubory (například soubory vzniklé během překladu v IDE, zkompilované soubory apod.) zahrnovat do commitů, bude to v každém dalším commitu obtěžovat a znepřehlední to celou historii. Dalším problémem generovaných souborů v repozitáři je jejich velikost, tím pádem zvýšení nároků na úložiště i snížení výkonu celého SCM řešení.

Funkčnost software – každý commit, který už je přístupný dalším lidem a systémům, by neměl vyvíjený software dostat do stavu, ve kterém by např. nešel zkompilovat. Ideálně by měl být daný software bez chyb. Dodržováním tohoto pravidla a kontrolování vlastních změn před odesláním se sníží počet chybových hlášení v dalších fázích.

Podobně jako v předchozí fázi i v této by měl stejný SCM systém sloužit jak vývojářům, tak i provozu.

4.3 Sestavení

Sestavení (build) je fáze, ve které vzniká spustitelná aplikace. Zde ve velké míře záleží na použité technologii a platformě. Při tvorbě webové aplikace v jazyce PHP bude sestavení vypadat jinak a obsahovat výrazně jiné kroky, než při sestavení desktopové aplikace napsané v jazyce C.

V zásadě ale existuje několik kroků, které se v různých obměnách aplikují při sestavení všech druhů aplikací. V rámci sestavení nedochází jen k překladu zdrojových kódů na spustitelné kódy, ale řeší se i získání a napojení závislostí, jako například knihoven třetích stran, databázových ovladačů a knihoven dané platformy, které jsou pro správnou funkci potřeba. V rámci sestavení také může docházet ke kopírování konfiguračních souborů, souborů pro tvorbu uživatelských rozhraní, obrázkových zdrojů, souborů s jazykovými překlady.

Informace o provedení nebo selhání sestavení software je jedna z prvních informací pro vývojáře o tom, že je aplikace v nepořádku. Z toho vyplývá, že fáze sestavení už částečně spadá i do fáze testování [20].

Důležité aspekty pro správně vytvořené sestavení dobře shrnuje Aiello [21], velké množství z těchto aspektů je důležité i pro praktiky DevOps. Pro kvalitně zpracovaný postup sestavení by měly platit následující body.

Jednoduchost – sestavení by mělo být tak jednoduché, jak jen to je možné, použití složitých konstrukcí a technologií na nesprávném místě může znamenat horší udržitelnost a rozšiřitelnost daného řešení. Spuštění sestavení by mělo být běžnou událostí na stisknutí „jednoho tlačítka“ nebo spuštění jednoho příkazu.

Opakovatelnost – pokud se proces sestavení spustí nad danými zdrojovými kódy opakovaně, výsledek by měl být stejný (nebo s minimálními změnami).

Rychlost a použitelnost – sestavení by mělo být spustitelné bez potřeby hlubších znalostí nástrojů, tak aby ho mohl spouštět každý vývojář v lokálním prostředí nebo nástroj k tomu určeným v centrálním prostředí.

Jedno sestavení pro více prostředí – správně vytvořené sestavení by mělo jít nasadit do všech prostředí, které se v rámci vývoje, testování a provozu vyskytují. Konfigurace, které se liší v různých prostředích (například přístup k databázi nebo cesta k souborům), by se měly objevit až v daném prostředí nebo při distribuci sestavení do daného prostředí.

Vysledovatelnost – z každého sestavení by mělo jít vysledovat, jaká verze zdrojových kódu byla použita, jaké požadavky se sestavením splnily aj.

Sestavení by mělo být takové, aby šlo používat pro vývoj i pro provoz v nezměněné podobě.

4.4 Testování

Poslední fází, do které většina vývojářů aktivně zasahuje, je testování. Testování software je velmi rozsáhlá a komplexní oblast. Konflikt mezi vývojáři a provozem zde nastává v potřebě vývoje rychle testovat nové funkčnosti a tlaku na provoz o co nejrychlejší nasazování nových verzí a přípravě nových testovacích prostředí.

V této fázi se v tradičních organizacích objevuje další početná a specializovaná skupina lidí zabývajících se čistě testováním. Vývojáři vidí většinou tyto lidi jako své nepřátele. Jednou z myšlenek DevOps je kromě spojení sil vývojářů a provozu i lepší komunikace právě s testery, kteří by měli být součástí vývojového týmu. Pokud nebudou vývojáři s testery aktivně komunikovat, bude mezi nimi vznikat propast, která se složitě překonává. Špatná komunikace vede vždy ke snížení kvality produktu, zvýšení jeho ceny a prodloužení doby dodání. Můžou nastat situace, kdy testeři testují funkčnosti, které už v současné verzi aplikace nejsou, ale protože nekomunikují s vývojem, tak se o této skutečnosti nedozvěděli [22].

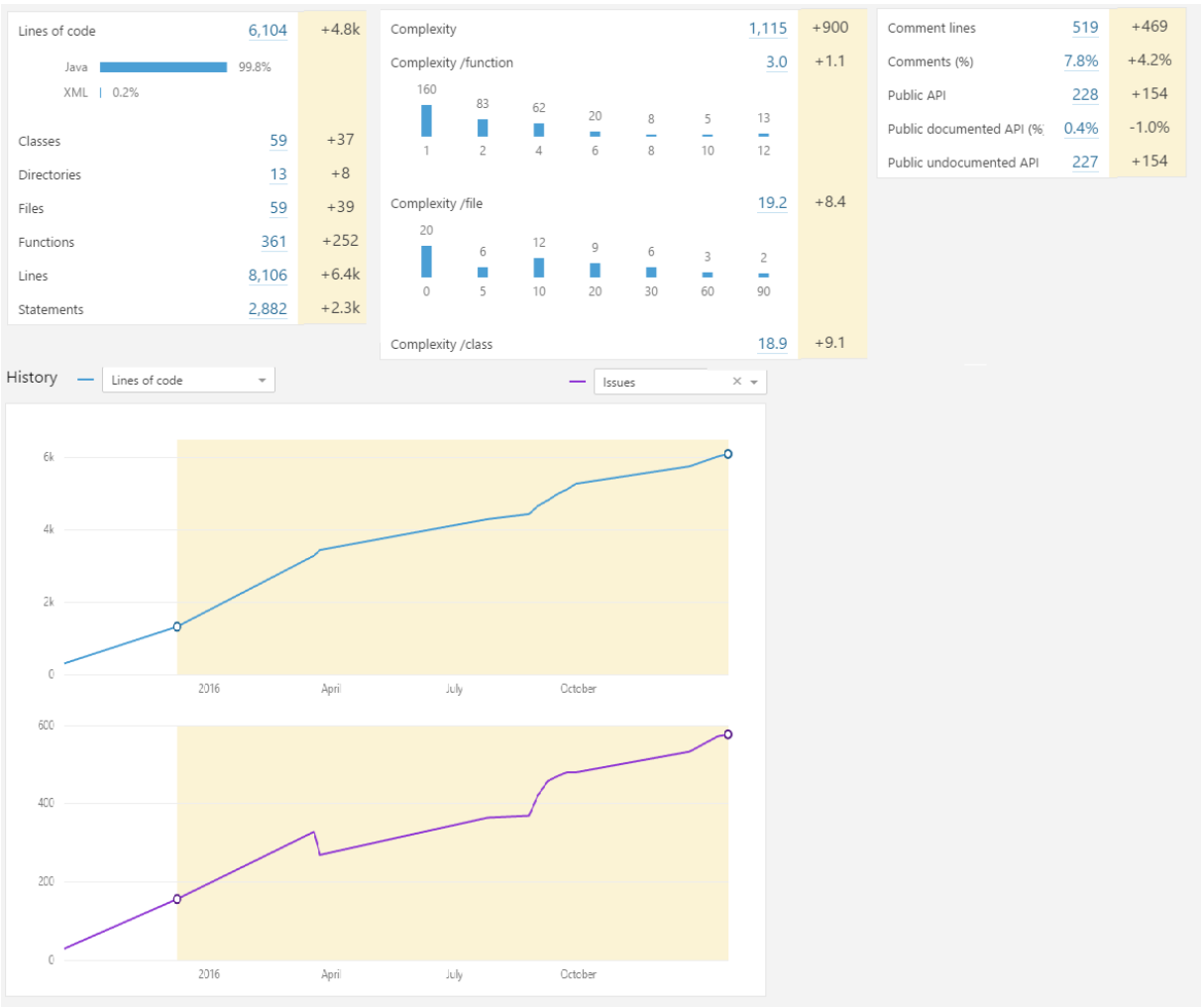
V rámci DevOps je kladen velký tlak na automatické testování a průběžnou kontrolu kvality, které mohou být snadno opakovatelné a rychle proveditelné. Vzhledem k používaným praktikám jako CI a CD (viz kapitola 3.4 Kontinuální integrace a 3.5 Kontinuální doručování) je kvalitní testování zcela zásadní. Bez testování je velmi složité se o CD pokoušet. Čím dříve se do testování zapojí i strana provozu, tím lépe pro celou aplikaci. Dojde tak ke dřívějšímu odhalení výkonnostních, bezpečnostních nebo integračních chyb [22].

Za formu testování se považuje i časté a opakující se provádění sestavení a nasazení softwaru „na zelené louce“. Kdy se začíná s úplně čistým prostředím, do kterého se uvolněný software

nasadí. Pokud se tato činnost děje pravidelně, tak se odhalí možné problémy mnohem dříve a ne až v kritický okamžik.

4.4.1 Statická analýza a kvalita kódu

Do testování se řadí i provádění statické analýzy. Statická analýza je analýza prováděná bez spouštění vyvíjené aplikace. Mnohé chyby je možné najít dříve než je software vůbec spustitelný, a tím ušetřit náklady na pozdější opravu. Ve fázi testování se může automaticky spouštět software, který projde poskytnuté kódy a vyhodnotí metriky kvality podle daného jazyka a nastavení. Výstup z takového software pak může vypadat podobně jako na obrázku 7, na kterém je výstup z programu SonarQube.



Obrázek 7: Ukázka výstupu statické analýzy kódu programu SonarQube. Několik metrik a porovnání v čase.

Mezi metriky používané během statické analýzy patří například:

- počty řádků, metod, objektů nebo souborů,

- provázanost mezi jednotlivými objekty,
- procentuální duplicita kódu,
- složitost funkcí, tříd,
- počet nevyužitých metod, proměnných,
- pokrytí testy.

Statická analýza se ale nemusí týkat jen kódu [22]. Do procesu kontroly lze zařadit například testování dokumentace nebo generovaných souborů (databázové skripty, instalační soubory, apod.). U dokumentace lze kontrolovat její formální správnost. U generovaných souborů například jejich počet a správné umístění.

Code review je technika, při které dochází ke kontrolování kódu před začleněním do ostatního už prověřeného kódu. Revizi kódu si ve většině případů vývojáři dělají mezi sebou. Touto technikou kromě snížení počtu chyb rostou vědomosti a znalosti v rámci týmu. Taky je tato technika ochranou před ztrátou znalosti kódu při odchodu pracovníků, protože každý kód znají alespoň dva lidé. Je to také ideální cesta pro vzdělávání juniorů, kteří jsou revidování zkušenými vývojáři a naopak můžou vykonávat review zkušenému vývojáři.

4.5 Uvolnění verze

Uvolnění verze (release) je fáze cyklu, která už je ve většině společností fází pro provozní oddělení. Uvolnění verze je činnost, při které se z daného sestavení stane artefakt vhodný pro nasazení do produkčního prostředí.

V rámci uvolnění verze se obvykle dělá několik činností. Mezi nejobvyklejší činnosti lze zařadit:

- označení daného sestavení unikátním identifikátorem,
- zanesení informace o sestavení do verzovacího systému,
- vytvoření seznamu změn,
- vytvoření balíčku pro uvolnění (vhodného pro distribuci),
- sběr všech artefaktů nutných pro nasazení nové verze,
- úprava uživatelských dokumentací,
- příprava prostředí pro novou verzi.

Než se nově vytvořená verze dostane do další fáze nasazení v produkčním prostředí, probíhá zpravidla intenzivní testování v prostředí předprodukčním [22].

Z pohledu DevOps je podobně jako u fáze sestavení kladen důraz na automatizování s možností rychlého opakování a snadné inicializace. Klade se také důraz na přehled o tom, ve kterém prostředí se momentálně nachází jaká verze aplikace. Více informací o automatizovaném a kontinuálním uvolňování je popsáno v kapitole 3.5 Kontinuální doručování.

4.6 Nasazení

Fáze, která je v tradičních společnostech s odděleným vývojem a provozem fází pro provozní týmy, se nazývá nasazení (deploy). Hlavním cílem je doručit aplikaci k zákazníkovi tak, aby ji mohl začít používat.

Při tvorbě jednoduchých aplikací se může jednat o operaci skládající se z jednoho nebo malého množství kroků, které se dají manuálně provádět jednoduše a bez chyb. S větší složitostí roste šance na lidskou chybu a celý proces trvá déle. Cílem CD (viz kapitola 3.5 Kontinuální doručování) je co možná nejčastější nasazení nových otestovaných a stabilních verzí. Pro časté nasazování je automatizace nezbytná. Protože s automatizací se snižuje riziko lidské chyby a celý proces nasazení může být mnohem rychlejší.

Nasazení do produkčního prostředí by mělo být co nejrychlejší a v ideálním případě bez omezení dostupnosti nasazované aplikace. Před nasazením do produkčního prostředí probíhá obvykle nasazení do prostředí předprodukčního, které by se mělo co možná nejvíce podobat prostředí produkčnímu (včetně aplikací, dat i hardwaru).

Postup nasazení silně závisí na používaných technologiích a prostředí, do kterého se daná aplikace nasazuje. V případě webové aplikace, která není kritická (může mít krátký výpadek), může typický průběh vypadat asi takto:

- příprava balíčku změn pro nasazení,
- kontrola aktivity uživatelů – vhodné časové plánování,
- nahrání změn,
- přepnutí systému do údržbového stavu (informace o údržbě, odhlášení uživatelů),
- vypnutí systému,
- aplikace změn,
- vymazání mezipaměti,
- zapnutí systému,
- kouřové testy (můžou sloužit i pro inicializaci mezipaměti),
- přepnutí do běžného stavu,
- záznam o nasazení verze do plánovacího systému.

Po nasazení je důležité více než jindy hlídat výskyt chybových hlášení a monitorovat zdroje systému. V případě problému je vhodné provést přechod zpět na starší verzi, případně uvolnit novou verzi s opravenou funkčností.

4.7 Provoz

Provoz (operate) aplikace je hlavním úkolem pro tým provozu. Oddělení provozu mívá odpovědnost za korektní provoz aplikace v produkčním prostředí. Opět jako v předchozí fázi i zde velmi záleží na povaze aplikace a technologiích, které ji tvoří. Jednoduchá webová aplikace fungující na sdíleném webovém hostingu nemá na provoz ze strany dodavatele aplikace v podstatě žádné

nároky. Úplně jiné nároky má aplikace běžící ve vlastním prostředí s integrací na další aplikace. Někde uprostřed leží běh aplikace v cloudovém prostředí, kdy jsou jednotlivé úkoly provozu rozprostřeny mezi dodavatele cloudového prostoru a provozovatele aplikace.

DevOps mezi cíle týmu provozu zařazuje také systémovou podporu a pomoc vývojářům.

Mezi činnosti, které se v části provoz vyskytují, patří:

- údržba a aktualizace operačních systémů,
- příprava a údržba prostředí pro běh aplikace,
- zálohování a obnovování systémů,
- správa síťového rozhraní a komunikace,
- bezpečnostní dohled,
- monitoring,
- škálování systémů.

4.8 Monitorování a záznam běhových informací

Poslední fází, kterou se DevOps zabývá, je monitorování aplikací a systémů. V produkčních prostředích bývá tato fáze plně v kompetenci oddělení provozu. Existuje spousta informací, které lze z monitorování provozu systému vytěžit. Od informačních záznamů z fungování aplikace, přes monitoring používaných funkcí v systému, až po vytížení procesoru nebo sítě.

Data z běžících systémů je vhodné nějakým způsobem zpracovávat a těžit z nich informace i v případě, že se nejedná o chybové hlášení nebo výkonnostní problémy. Užitečnou informací může být i počet využití konkrétních funkcí systému, časové období nejvyšší zátěže systému apod.

Monitorování v době, kdy je systém v režimu údržby, může sloužit k lepšímu odhadnutí nejvhodnější doby pro údržbu nebo může poukázat na nutnost zprovoznit systému v bezvýpadkovém režimu.

Pro DevOps je nesmírně důležité, aby se informace získané z provozu systému (logy) dostaly zpátky k vývojáři. Stejně tak důležité je, aby měli vývojáři dostupné informace o vytížení zdrojů aplikace a využití funkcionalit. Tato zpětná vazba povede ke kvalitnější a výkonnější aplikaci. Pokud se vývojáři nemají jak dostat k informacím například o změně spotřeby paměti s novou verzí systému, mohou nabýt dojmu, že je aplikace stále stejně funkční jako v předchozí verzi, i když oddělení provozu může mít s využitím paměti problém. Je obvyklé, že provoz posílá informace pouze v případě problému. Ale tento způsob předávání informací může fungovat jako úzké hrdlo, které způsobí prodloužení doby opravy problému. Následkem delší doby opravy problémů může být ztráta důvěry uživatele nebo smluvní pokuta.

S daty vyprodukovanými běžícím systémem by se ale mělo zacházet opatrně. I přes důslednou kontrolu se do nich můžou dostat citlivé informace, takže je vhodné s nimi pracovat podobně jako se zálohami aplikačních dat. Data je také vhodné po nezbytně nutnou dobu archivovat,

aby je bylo možné v případě potřeby použít ke zpětné analýze problému nebo kontrole výkonu. Doba archivace opět silně závisí na povaze aplikace.

Existuje velké množství dat, které lze sledovat a nějakým způsobem dále vyhodnocovat. Pro každé použití se hodí využití jiných sestav dat a jejich zdrojů. Mezi základními daty, která lze sledovat, jsou data z následujících komponent.

Aplikace – programátory definované výstupy na různých úrovních (info, warn, error, ...).
Aplikační server – stav paměti, počet a využití vláken.
Databázový server – počty aktivních připojení, informace o pomalých dotazech, stavy indexů.
Hardware serveru – využití operační paměti, využití procesoru, pevného disku.
Data o síťovém provozu – toky sítí, počty přenesených bajtů.

5 Analýza současného stavu, popis provedených změn a nástrojů ve firemním vývojovém procesu

Druhou částí této diplomové práce je praktické využití získaných vědomostí a dovedností z kapitol výše v reálném prostředí malé firmy. V následujícím textu popíšu klíčové oblasti vývoje software z konkrétní firmy, v textu bude nastíněna jak existující konfigurace a nastavení procesů, tak provedené vylepšení, úpravy a využití nových postupů a nástrojů.

I přes maximální vyvinuté úsilí nebylo možné všechny úpravy dokončit ve stanovém termínu této diplomové práce. V některých částech jsou proto rozepsány možnosti dalšího postupu, kterým se budu věnovat dále po dokončení této práce.

5.1 Identifikace firmy a jejích produktů

Jako firmu, ve které prakticky vyzkouším získané vědomosti, jsem využil firmu VRK plus s.r.o., kde jsem už několik let zaměstnaný. V této firmě jsem vystřídal již několik pozic a zaměření, mezi kterými je i správa infrastruktury a zajišťování prostředí pro běh aplikací na testovacích i produkčních serverech.

Firma VRK plus s.r.o je malá firma s 10 zaměstnanci, která se zabývá vývojem software v několika oblastech. Mezi hlavní oblast patří vzdělávací instituce v České republice a na Slovensku. Firma se dále zabývá vývojem software pro správu a evidenci členské základny klubů a vývojem volnočasového portálu [23].

Na hlavním vývoji se v tuto chvíli podílí 6 programátorů přímo zaměstnaných a několik externistů. Většina programátorů v jednom čase pracuje na vývoji pouze jednoho produktu.

Následující kapitoly čerpají z vlastních poznatků a pozorování a z popisu portfolia firmy [23].

5.1.1 Problémy ve firemní kultuře

Ve firmě není žádný člověk, který by dělal výhradně správu infrastruktury, zdrojů a provozu obecně (Ops). Většina vývojářů dostávala určité provozní úkoly už od počátku. Jeden vývojář byl vždy zodpovědný za správný běh serveru jako celku. Vzhledem k tomu, že samotní vývojáři brali administrátorskou práci většinou jako nutné zlo, vypěstovali si k ní určitý odpor a zároveň z ní pociťovali strach. Jinými slovy byla pro ně administrátorská práce prací, která je pouze omezovala v tom psát aplikační kód a vyvíjet.

U některých aplikací neexistoval žádný konkrétní postup nasazení nové verze, pouze se předával mezi vývojáři, kteří měli daný produkt v danou chvíli pod správou.

Vzhledem k důvodům nastíněným v předchozích odstavcích padlo významné rozhodnutí vyzkoušet praktiky DevOps, které dokážou manuální práci správců odstranit, automatizovat a zjednodušit. Současně díky prozkoumávání technologií a postupů se programátoři mohou dostat k jinému pohledu na vývoj, který jim může pomoct i v programování a vlastním vývoji software.

Další změnou, kterou by mělo přinést zavádění technik DevOps, je větší jistota při úpravách konfigurací a přípravě prostředí. S rozvojem firmy se z práce okolo provozu serverů nestane věc, která by celý vývoj zpomalovala.

Pokud jde o ekonomickou stránku věci, tak s dlouhodobého pohledu jsou určitě techniky DevOps perspektivnější než zaměstnání člověka s vyhraněnou správcovskou pozicí. V případě potřeby specialisty je možné najmout člověka na konkrétní úkony nebo proškolení zaměstnanců.

5.1.2 Produkty a vyvíjený software

Následující kapitoly využívají příklady s konkrétními a reálnými aplikacemi, proto tato kapitola popíše hlavní vyvíjené produkty a aplikace.

Firma VRK plus s.r.o. v tuto chvíli vyvíjí 3 architektonicky odlišné programy pro podporu školství, 2 aplikace pro práci s členskou základnou a několik tematických webových portálů.

Internetový školní informační systém - ISIS Integrovaný informační systém pokrývá většinu činností spojených s řízením vysoké nebo vyšší odborné školy v České republice. Systém má několik různých modulů a částí. Je napsaný v jazyce Java a typově se jedná o aplikaci dodávanou zákazníkovi, který se stará o přípravu prostředí na úrovni operačního systému (aktualizace, zálohy, firewall). Příprava běhového prostředí (Java, Tomcat, samotná aplikace) a databáze je už záležitostí firmy.

Systém je aktuálně provozován u tří zákazníků, kteří mají uzpůsobenou aplikaci pro své konkrétní potřeby.

Program pro práci se školními vzdělávacími programy - SMILE Další software vyvíjený firmou VRK plus s.r.o. je software pro vytváření, editaci a publikování školních vzdělávacích programů a podporu výuky. Jedná se o klasickou desktopovou aplikaci vyvíjenou nad platformu Eclipse v jazyce Java. Aplikace může fungovat jak v lokální verzi (s lokální souborovou databází) tak ve verzi síťové (nejčastěji s využitím relační databáze MySQL). Instalaci i běh aplikace si zajišťují zákaznici sami, ale mohou využívat firemní podporu.

Projekt Elektronizace vzdělávacího systému regionálního školství Slovenské republiky V rámci tohoto projektu firma vyvíjí software pro práci se školními vzděláváními programy. Software je napsaný v Javě pod platformou Liferay. Firma v tomto projektu figuruje jako subdodavatel konkrétní části celého systému. Systém je typickou portálovou aplikací, hostovanou v datacentru ministerstva školství Slovenské republiky.

Během tohoto projektu se jasně ukázalo, jak vypadá nespolupráce a vzájemná nekomunikace mezi vývojáři a administrátory, kdy docházelo (a dochází) k absurdním situacím, ke kterým by ve správně fungující softwarové firmě docházet nemělo.

Editor Státních vzdělávacích programů V rámci projektu uvedeného v kapitole výše vzniká také program pro přípravu státních vzdělávacích programů. Tento software funguje jako desktopová aplikace s centrálním serverem, který poskytuje data a řídí aktualizace. Software je napsaný v jazyce Java a využívá JNLP[7].

Aplikace pro klubovou evidenci Klub a Evidence Další vyvíjenou aplikací je aplikace pro klubovou evidenci, která je napsaná v jazyce Java s použitím technologie GWT[8]. Aplikace je typickou JavaEE aplikací bežící na Tomcat serveru a ovládanou z webového prohlížeče. Aplikace obsahuje integraci do webového portálu pohodlne.info (viz následující kapitola). Portál pohodlne.info by měl časem celou aplikaci v rámci zužování technologií a sjednocování nahradit.

Portál pohodlne.info Informační portál pohodlne.info sloužící veřejnosti pro získávání informací o pořádaných (převážně sportovních) událostech v jejich okolí, má přímé napojení na klubovou evidenci. Portál je postavený v jazyce Java nad platformou Liferay.

Portál slouží jako centrální bod, na který dále navazují další tematicky zaměřené portály.

Portál detske-tabory.info Portál pro evidenci dětských táborů, který postupně přebírá funkčnosti dosluhující a špatně udržitelné stejnojmenné PHP aplikace. Portál poskytuje návštěvníkům informace o táborech v jejich okolí a pořadatelům možnost napojení na evidenci členů.

Portál ceske-skoly.info Portál ceske-skoly.info je další portál z rodiny portálů pohodlne.info, který má napojení na výše zmíněný program SMILE a slouží převážně učitelům středních a základních škol pro převod školních vzdělávacích programů do systému České školní inspekce.

5.2 Úpravy v systému pro plánování

Firma v roce 2015 migrovala ze systému Mantis do systému Redmine, který je hostovaný na vývojovém serveru firmy. Je přístupný snadno zapamatovatelnou URL adresou. Úkoly se člení do projektů podle vyvíjených produktů. Do systému mají přes individuální účty přístup všichni zaměstnanci, externích spolupracovníci i někteří zákazníci.

Kromě systému Redmine se pro plánování a zadávání úkolů pro projekty, které jsou v počátku vývoje a je tak třeba zadat velké množství funkčností v prvních etapách, využívá i Google Docs, který umožňuje rychlé zadávání a snadnou editaci. Využití Google Docs ale přináší několik problémů:

- úkoly se špatně historizují a zpět trasují,
- úkol nelze efektivně přiřazovat jednotlivým řešitelům,

[7] JNLP – Java Network Launch Protocol – technologie pro získání, instalaci a aktualizaci Java aplikace
[8] GWT – Google Web Toolkit – framework pro psaní webových aplikací

- složitější integrace do dalších nástrojů,
- větší množství úkolů se stává méně přehledné,
- omezená práce se stavy úkolu,
- neexistující notifikace o změně stavu úkolu.

Někteří zaměstnanci dále využívají vlastní systémy poznámek pro přehled o denní úkolech apod. Tyto poznámky nejsou žádným způsobem formalizované a slouží pouze pro přehled jednotlivců.

5.2.1 Problémy a navrhované řešení

Problém je ve využívání Google Docs – systém nepodporuje základní požadavky na použitelný a udržitelný plánovací systém. Úkoly jsou na více místech, vznikají duplicity a nedodržuje se formát. Dalším problémem je nedodržování předepsaných přechodů mezi stavy.

Denní přehledy jednotlivců nejsou problémem. Úkoly (nebo odkazy) na úkoly v těchto „úkolníčcích“ jsou krátkodobého charakteru a slouží jen pro orientaci jednotlivých pracovníků.

Využívání systému Google Docs Během analýzy a hloubkových rozhovorů jsem odhalil hlavní příčinu využívání Google Docs namísto systému Redmine. Problém se vyskytuje hlavně u úkolů, které vznikly po manuálním systémovém testování a v úvodní fázi vývoje. Problém tkví v nemožnosti rychle zadávat do systému Redmine nové úkoly a neztrácet tak koncentraci například při prováděných testech. Pokud se tester při vykonávání testů zastaví, aby si poznamenal úkol, tak v systému Redmine s tím stráví (i přes relativně malý počet povinných polí) mnohem více času než v seznamu Google Docs. Pokud by poznámky během testování nevznikaly hned, velké množství by se ztratilo a nebyly by zaznamenány vůbec. Následné manuální přepisování do systému Redmine by stálo spoustu času navíc.

Po firemní poradě jsem dospěl ke třem možným řešením problému. Prvním řešením je migrace ze systému Redmine do jiného systému, který by umožňoval rychlejší zadávání, ale současně splňoval další podmínky pro plánovací systém. Mezi vhodné kandidáty patří například systém Trello. Další migrace ale znamená změnu postupů práce množství lidí.

Druhé možné řešení je využití rozšíření pro hromadné vkládání úkolů ze souboru[9]. Toto řešení sice přináší trochu práce navíc pro zadávajícího, ale ostatních se změna nedotkne a při použití automatického skriptu pro převod dat do importovatelné podoby to ani zadávajícímu tolik práce navíc nepřinese.

Posledním možným řešením je využití rozšíření pro hromadné vytváření. Po analýze dostupnosti takového rozšíření jsem ale došel k názoru, že takové rozšíření neexistuje a bylo by ho třeba na míru vytvořit.

[9]Například - `http://www.redmine.org/plugins/redmine_import_issues`

Po zhodnocení všech možností jsme se rozhodli dále pokračovat v systému Redmine a omezit využívání Google Docs na co nejmenší možnou míru. V okamžiku nižšího množství práce na projektech pro zákazníky bude vyvíjeno rozšíření pro hromadné zadávání.

Nedodržování přechodu mezi stavy V systému Redmine je nadefinováno několik základních stavů, ve kterých se zadaná funkcionalita může nacházet. Přechody i stavy samotné byly definované ve firemní dokumentaci. Stavy jsem zrevidoval a nastavil jejich význam tak, aby všichni zaměstnanci znali jejich smysl.

Kromě ruční změny stavů jsem do procesu vytváření a nasazování aplikací zařadil automatickou změnu stavu. Více v kapitole 5.4.1 Jenkins – nástroj pro kontinuální integraci.

5.3 Práce se systémem pro správu zdrojových kódu

Ve firmě se od roku 2015 využívá namísto systému CVS[10] systém Git. Centrální repozitáře jsou umístěny na vývojovém serveru firmy. Jednotlivé repozitáře jsou na vývojovém serveru spravovány prostřednictvím aplikace Gitolite, která řídí vytváření a přístup jednotlivých uživatelů do konkrétních repozitářů.

Systém Git je aktivně používán všemi vývojáři. Určitým problémem je absence jednoduchého a bez technických znalostí přístupného rozhraní, ve kterém by byl přehled o veškerých aktivitách vývojářů napříč repozitáři. Pro aplikace Gitolite existuje rozšíření, které přidává webové rozhraní, ale jeho schopnosti jsou značně omezené. Pro získání těchto informací o aktivitě uživatelů je nutné repozitáře naklonovat a informace vyčíst z logů.

Tento způsob omezuje získání těchto informací v podstatě jen na vývojáře. Pokud chce tyto informace někdo z managementu, je to pro něj složité. DevOps ale využívá praktiky sdílení informací, a proto je dobré, aby k těmto informacím měli přístup všichni. Vhodnou náhradou systému Gitolite by mohl být například systém GitHub nebo GitLab.

Při trvání na požadavku na umístění zdrojových kódu na vlastním serveru a pokud možno nízké pořizovací a provozní náklady vychází nejlépe systém GitLab.

5.3.1 GitLab

GitLab je systém pro správu repozitářů Git, napsaný v jazyce Ruby. Kromě možnosti provozu na serverech společnosti GitLab umožňuje provoz v bezplatné variantě i na vlastím serveru. Standardní distribuce je přes balíčkovací systém daného operačního systému (v oficiálních variantách jsou všechny obvyklé linuxové distribuce). Existují také další postupy pro instalaci prostřednictvím systému Puppet, Vagrant nebo Docker [24].

Systém GitLab neposkytuje pouze úložiště pro Git repozitáře, ale jedná se o komplexní nástroj, který dokáže zabezpečit vše od zaznamenávání úkolů po automatické nasazení. Nicméně jeho možnosti v zadávání úkolů jsou omezené, tento systém je vhodné využit u menších projektů.

[10] CVS – Concurrent Version System, nástroj pro správu zdrojových kódů

Na rozdíl od systému Redmine neobsahuje možnost vytváření vlastních atributů u jednotlivých úkolů. Dále obsahuje i systém pro kontinuální integraci a sestavování prostřednictvím subsystému GitLab CI a poskytuje podobné možnosti jako úlohy v systému Jenkins, ale Jenkins má mnohem větší základnu uživatelů a rozšíření. Další výhodou systému GitLab je možnost použití mobilní aplikace a možnost řešení drobných změn odkudkoliv.

K praktickému nasazení aplikace GitLab na místo existujícího Gitolite řešení během práce na této diplomové práce nedošlo, nicméně je to v plánu při dalším pokračování. Samotný přechod nebude pro vývojáře představovat žádnou změnu, jen přibude rozhraní, které bude dostupné i pro lidi bez znalosti fungování systému Git. Další možností bude konfigurace jednoduchých úloh v rámci CI a CD do GitLab CI.

5.3.2 Rozdělení repozitářů

V době, kdy se ve firmě využíval systém CVS, existoval v podstatě jen jeden repozitář, ze kterého vznikalo více aplikací – SMILE, ISIS i Klub. Při přechodu na systém Git se tento repozitář nechal ve stejném duchu spojení zdrojových kódů aplikací dohromady.

Pro různé verze aplikací, které jsou podporované, se používají větve, které tak jsou trvalé. Některé větve slouží pro změny, které by byly nekompatibilní s jinou aplikací. Aplikace sice využívají společný základ, ale funkčně i architektonicky jsou zcela odlišné. Ideální by bylo tyto aplikace rozdělit tak, aby každá měla svůj vlastní repozitář, a pro sdílení malého množství kódu využívaly například submoduly nebo sestavovací závislost.

Pro aplikaci pohodlne.info, která se skládá z několika subprojektů, opět existuje jeden repozitář, ve kterém jsou všechny dílčí projekty. Výsledkem je jedna webová aplikace. Zde je ale situace jiná, protože ze všech subprojektů vzniká ve finále jedna aplikace, proto je shromáždění zdrojových kódů na jednom místě naopak žádoucí.

Při analýze těchto problémů jsem došel k několika variantám rozdělení zdrojových kódu do repozitářů.

Monolitický repozitář je nejjednodušší způsob. Je to repozitář, ve kterém jsou uložené veškeré zdrojové kódy všech aplikací. Je to nejtriviálnější cesta, jaká existuje, která se ale s velkým množstvím aplikací může stát nevhodnou a brzdící. Nicméně existují případy [25][26] známých firem, které používají (nebo používaly) tento způsob zcela funkčně a s úspěchem.

Výhodou je snadná změna jedné funkce napříč všemi projekty, možnost jednoduše definovat závislosti mezi projekty – všichni využívají stejnou souborovou strukturu, a tak lze odkazovat přímo.

Ale při opravdu velkých projektech s dlouho historií může nastat problém s výkonem základních operací. Složitější je řešení případů, kdy je nutné poskytnout určitým lidem jen část repozitáře. Problém taky může nastat ve složitějším nastavení CI, tak aby se sestavovala vždy jen část, která je pro danou komponentu nutná.

Samostatné repozitáře pro jednotlivé projekty, kdy platí že, co jeden projekt, to samostatný repozitář. Při tomto řešení se snadno v jednotlivých repozitářích hlídají změny. V případě sdílených programových komponent a zdrojových kódů je možné využít v případě systému Git například submoduly, které umožní vložit jeden repozitář do druhého. Submoduly ale na druhou stranu přináší složitější práci pro vývojáře (musí se provést commit do subrepozitáře i do hlavního repozitáře), problematické může být i větvení. Snadnější je nastavení CI pro sestavení konkrétních částí.

Závěrem je, že nejvhodnější možností pro firmu je použít kombinaci obou řešení. Cesta čistě monolitického repozitáře se ukázala v případě repozitáře pro SMILE, ISIS a Klub jako nevhodná, z důvodu fakticky jiných požadavků a odlišného tempa vývoje tří výše zmíněných aplikací. Pokud jde o aplikaci pohodle.info, tak navrhovaná možnost využít pro každý modul samostatný repozitář se ukázala jako nevhodná a na celý repozitář a všechny moduly se bude nahlížet jako na kompletní aplikaci.

5.3.3 Využití větví pro vyvíjené vlastnosti

Využití větví pro vyvíjené vlastnosti (Feature branch) je technika, kdy se jednotlivé dílčí úkoly vytvářejí v samostatných větvích, které jsou publikovány a následně integrovány do hlavní větve. V oblasti open-source je tento způsob vývoje vyžadovaný, protože ke změnám dochází relativně pomalu.

Ve firmě se tento systém při běžném vývoji nepoužívá a všechny změny se rovnou aplikují do hlavní větve, takže dochází k pravidelné a nepřetržité integraci. Při používání větví pro jednotlivé funkčnosti hrozí riziko velkého slučování změn, kdy se v různých větvích změní stejné soubory, které se nakonec musí spojit dohromady [27].

Nicméně v projektu EditoruŠVP se této možnosti využívá pro kontrolu a revizi kódu nového vývojáře. Všechny jeho změny kontroluje zkušený vývojář, který mu tak dává zpětnou vazbu a příležitost ke zlepšení. Současně se nestane, že by se do hlavní větve dostal kód, který není funkční nebo nesplňuje dané požadavky.

5.4 Automatizace sestavení a nasazování

Automatizované sestavení a nasazování je jednou z klíčových oblastí DevOps. Proto to bylo jedno z míst, na které jsem se v praktické části této práce zaměřil. Tyto techniky jsou více rozpracovány v kapitole 3.5 Kontinuální doručování. Ve firmě se používá několik různých způsobů pro vytvoření sestavení a jeho doručení do cílového prostředí.

Aplikace ISIS, SMILE a Klub jsou sestavované pomocí Ant skriptu uloženého na vývojovém serveru, který je spouštěn manuálně přes konzoli na daném serveru. Uložení Ant skriptu jen na

vývojovém serveru je problematické z toho důvodu, že nelze stejné sestavení spustit snadno i na lokálním počítači pro otestování funkčnosti.

Moduly do portálu pohodlne.info se všechny sestavují pomocí nástroje Maven, ale sestavení se spouštělo jen na lokálních stanicích vývojářů, první modul měl i Bash skript na vývojovém serveru.

Dalším problémem je nasazování a distribuce těchto aplikací. Aplikace ISIS a Klub se po sestavení do `.war` souboru manuálně nahrávají prostřednictvím SCP protokolu na testovací i produkční servery. Pokud jde o aplikaci SMILE, která je aplikací desktopovou, dochází k nahrání nového balíčku pro instalaci přes protokol FTP do určené složky ve webové prezentaci firmy, odkud si jej mohou zákazníci stáhnout a nainstalovat.

Testy před a po sestavení, případně po nasazení do testovacího prostředí, jsou prováděny jen manuálně. Rozšíření o automatické testování by bylo vzhledem k postupu sestavení a nasazení dalším manuálním a složitým krokem.

Doba mezi odesláním commitu a nasazením Problém je také ve zbytečně dlouhé době mezi odesláním commitu s vyřešeným úkolem a nasazením aplikace s touto změnou do testovacího (resp. produkčního) prostředí . K sestavení a nasazení do testovacího (resp. produkčního) prostředí dochází nepravidelně a vždy to znamená pro jednoho určeného vývojáře zastavení práce, spuštění sestavení změněných modulů, přenos sestavených modulů na testovací server, jejich aplikování a kontroly funkčnosti. To znamená spoustu manuální a opakující se práce, silně náchylné na chybu. Navíc pokud se daný vývojář nemohl této činnosti věnovat (z důvodu dovolené, nepřítomnosti, ...), tak sestavení stálo a nikdo se mu nechtěl věnovat a převzít ho.

Na obrázku 8 jsou ilustrovány akce v repozitáři aplikace pohodlne.info. Jednotlivé dny, ve kterých byl odeslaný alespoň jeden commit, jsou podtržené. Šedě zvýrazněné dny jsou dny, ve kterých se provádělo nasazení do testovacího nebo produkčního prostředí. Dny, které jsou zvýrazněny šedě s tučným písmem, jsou dny, kdy došlo k nasazení kompletně všech modulů aplikace. Netučné dny znamenají nasazení jen jednoho modulu.

Další obrázek 9 zobrazuje dobu prodlevy mezi commitem a nasazením do testovacího resp. produkčního prostředí. Na vodorovné ose je počet dnů prodlevy, na svislé ose je zobrazena četnost commitů. Z grafu je patrné, že nejvíce commitů bylo nasazeno do testovacího prostředí během 24 hodin. Nicméně existuje také 5 commitů, které čekaly na nasazení do testovacího prostředí přes třicet dní. Průměrná doba mezi commitem a nasazením do testovacího prostředí je asi čtyři dny.

Velký počet rychle nasazených commitů do testovacího prostředí může být podezřelý, protože se může jednat o commit, který byl u vývojáře déle, ale byl odevzdán až v den samotného nasazení, a došlo tak k prodloužení doby bez integrace se zbytkem týmu.

srpen '16

po	út	st	čt	pá	so	ne
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

září '16

po	út	st	čt	pá	so	ne
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

říjen '16

po	út	st	čt	pá	so	ne
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

listopad '16

po	út	st	čt	pá	so	ne
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

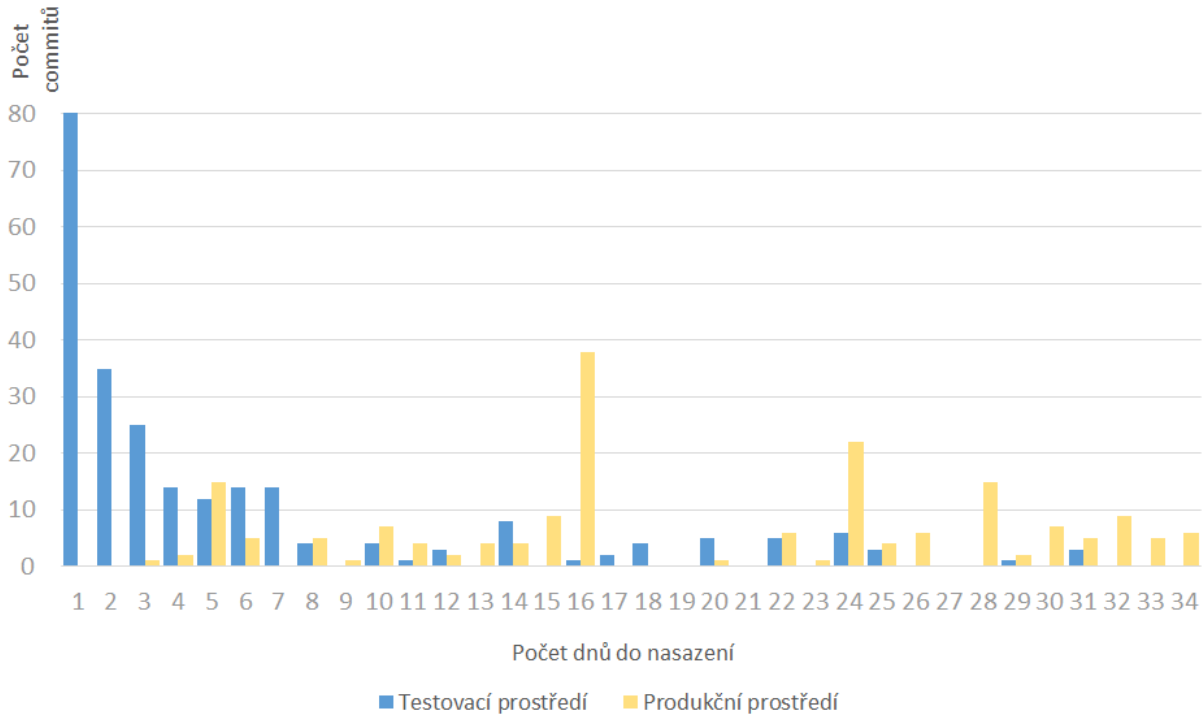
prosinec '16

po	út	st	čt	pá	so	ne
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

leden '17

po	út	st	čt	pá	so	ne
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Obrázek 8: Ilustrace dnů, ve kterých byl odeslaný commit a ve kterých bylo spuštěno sestavení



Obrázek 9: Graf počtu commitů dle doby do nasazení do testovacího resp. produkčního prostředí

Z obou těchto ilustrací je zřejmé, že doba mezi odesláním změny a nasazením alespoň do testovacího prostředí je u některých případů zbytečně dlouhá a testeři nemohou svou práci dělat kontinuálně.

Z důvodu nesourodosti, složitosti a manuální práce jsem se rozhodl pro využití automatizačního nástroje. Nástrojů, které zvládnou spustit sestavovací skript, provést případné testování a výsledné artefakty nasadit do daného prostředí, je velké množství a jejich výběr většinou závisí na použitých technologiích a zkušenostech.

Další kapitola popisuje nástroj Jenkins, pro který jsem se rozhodl z důvodu velké komunitní podpory, velkého množství rozšíření a relativně jednoduchého fungování.

5.4.1 Jenkins – nástroj pro kontinuální integraci

Jenkins je projekt s otevřeným zdrojovým kódem napsaný v jazyce Java. Vznikl odštěpením od nástroje Hudson v roce 2010. Jeho hlavním úkolem je spouštět nadefinovaného úkoly. Úkoly lze spouštět manuálně nebo na základě nějakého podnětu (čas, změna ve zdrojových kódech, externí volání). Tyto úkoly se můžou skládat z mnoha částí, od získání změn ve zdrojových kódech, přes provedení sestavení, spuštění testů, spuštění externích nástrojů (například pro statickou analýzu kódu), až po nahrání do repozitáře binárních souborů nebo přímo do běhového prostředí aplikace. Pro Jenkins existuje velké množství rozšíření [24].

Mezi jiné nástroje, které plní stejnou funkci, patří například placený Atlassian Bamboo, GitLab CI, Travis CI a další. Jenkins jsem vybral z důvodu jeho rozšířenosti a oblíbenosti. Dále se pyšní velkou uživatelskou komunitou[11]. Existuje pro něj široké množství rozšíření, což ale patří i mezi jeho nevýhody, protože se vyvíjejí rozdílným tempem a objevuje se nekompatibilita, různé způsoby nastavování apod. Další nevýhodou je složitější pokročilá konfigurovatelnost úloh (od verze 2.0 výrazně lepší) a vyšší nároky na výpočetní zdroje.

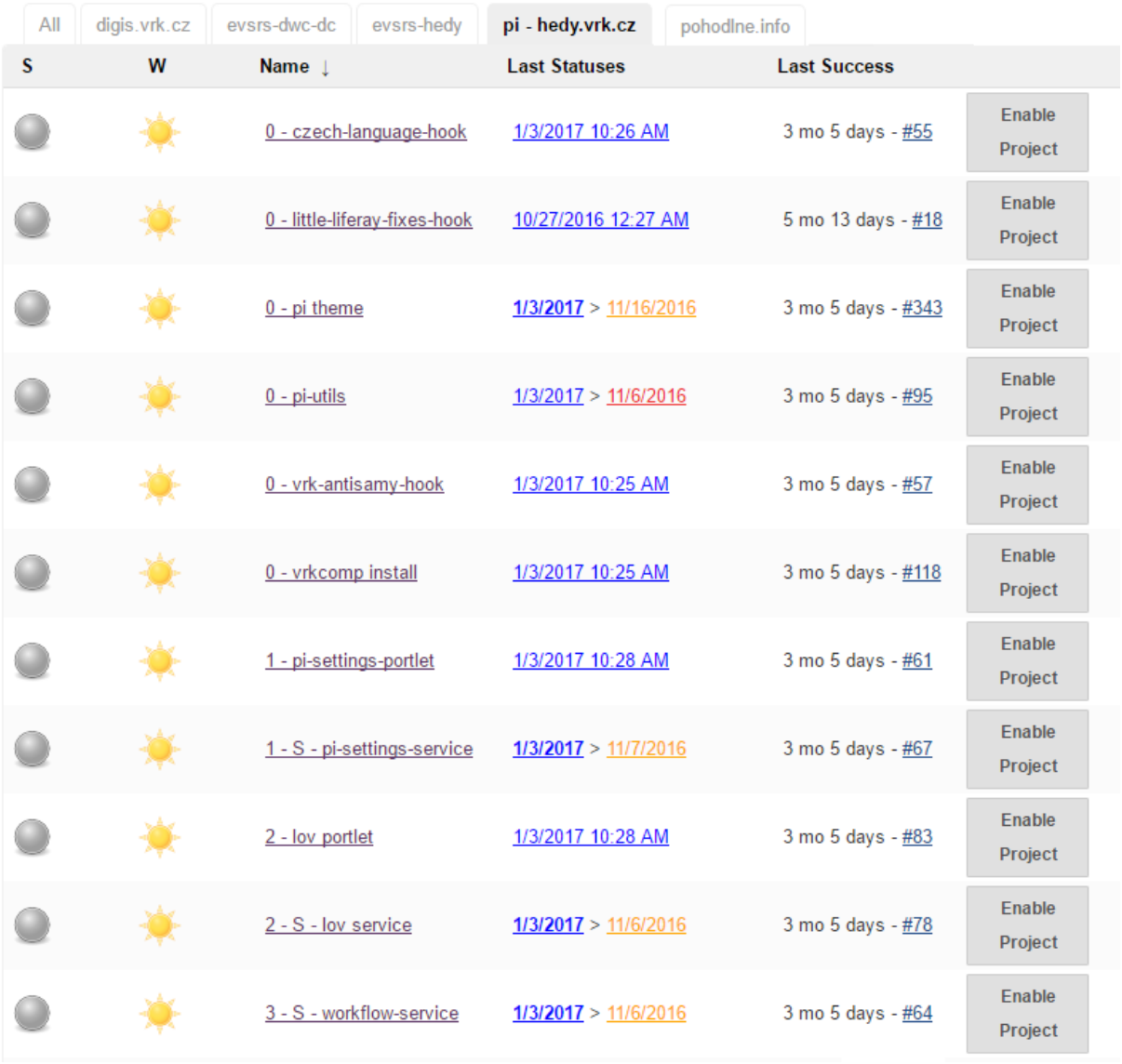
První verze úloh První verze Jenkins serveru, kterou jsem na vývojový server firmy nainstaloval a začal konfigurovat, byla verze číslo 1.634. Jako způsob instalace jsem zvolil instalaci prostřednictvím Bitnami stack[12].

Projekt, který byl ve firmě aktuálně nejživější a potřeboval intenzivní sestavení a nasazování, byl vývoj komponent pro aplikaci pohodlne.info. Vytvořil jsem úlohy pro jednotlivé moduly – viz obrázek 10. Vzhledem k tomu, že ve verzi Liferay, pro který jsou dané moduly vyvíjeny, je jediný způsob sdílení servisních rozhraní pro přístup k datům skrz různé moduly přes `.jar` soubory umístěné ve společném knihovním prostoru aplikačního serveru, vytvořil jsem vždy

[11] Na serveru Stackoverflow v době psaní této diplomové práce bylo více než dvacet tisíc otázek na téma Jenkins – `http://stackoverflow.com/unanswered/tagged/jenkis`

[12] Bitnami stack je instalátor, který obsahuje vždy všechny potřebné komponenty pro nativní instalaci daného softwaru. Umožní tak jeho rychlou a bezproblémovou instalaci na různých operačních systémech. Umožňuje také snadné spouštění u poskytovatelů cloudu.

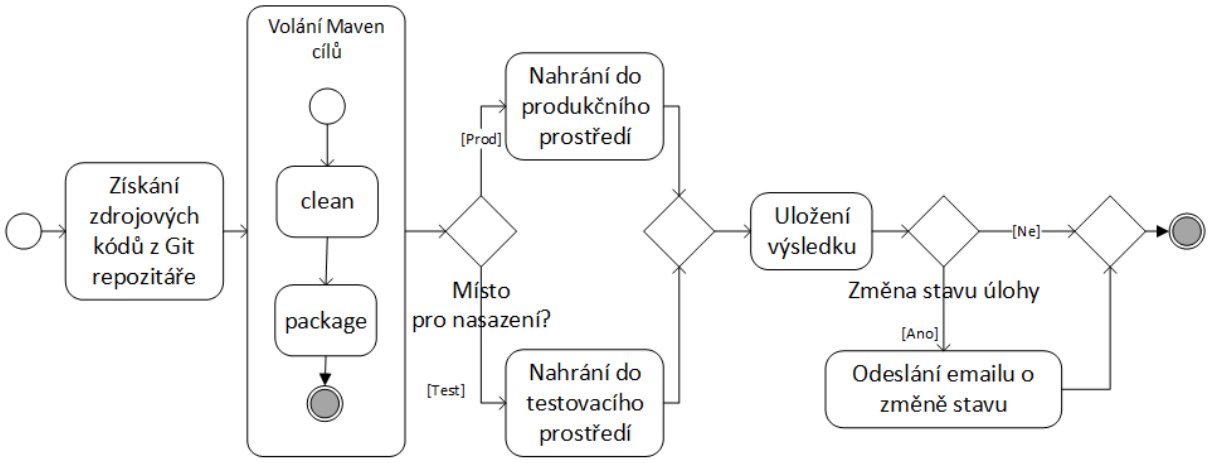
zvlášť úlohu pro modul (teoreticky lze nasazovat za běhu aplikačního serveru) a servisní knihovnu (po nasazení je potřeba server restartovat).



Obrázek 10: Ukázka několika úloh v Jenkins, pro sestavení a nasazení jednotlivých modulů do testovacího nebo produkčního prostředí

S postupně se zvětšujícím počtem modulů přibývaly i úlohy. Současně se projekt dostal do stavu, kdy se začal nasazovat i do produkčního prostředí. Proto jsem rozšířil každou úlohu o možnost určující, do jakého prostředí se bude po sestavení daného modulu nasazovat. Diagram celé úlohy je zobrazen na obrázku 11.

Poslední úloha, kterou jsem v první fázi vytvořil, byla úloha, která spouštěla postupně všechny úlohy jednotlivě za sebou. Spuštěním této úlohy se provedlo kompletní sestavení a nasazení všech modulů. V této úloze se využil paralelní běh těch úloh, které k sobě neměly



Obrázek 11: Standardní průběh jedné Jenkins úlohy pro sestavení a nasazení modulu do aplikace pohodlne.info

vztah a nezáleželo na pořadí sestavení (například modul pro vzhled a modul s překlady Liferay).

Změna stavu v Redmine po sestavení/nasazení Později jsme společně s celým týmem vývojářů a testerů došli k potřebě využít spouštění úloh i pro oznámení do systému pro zadávání úkolů. Používaný systém pro zadávání úkolu – Redmine, podporuje komunikaci s ostatními systémy přes HTTP API. Obě aplikace obsahují rozšíření pro integraci s tou druhou, ale žádná neobsahuje požadovanou funkčnost.

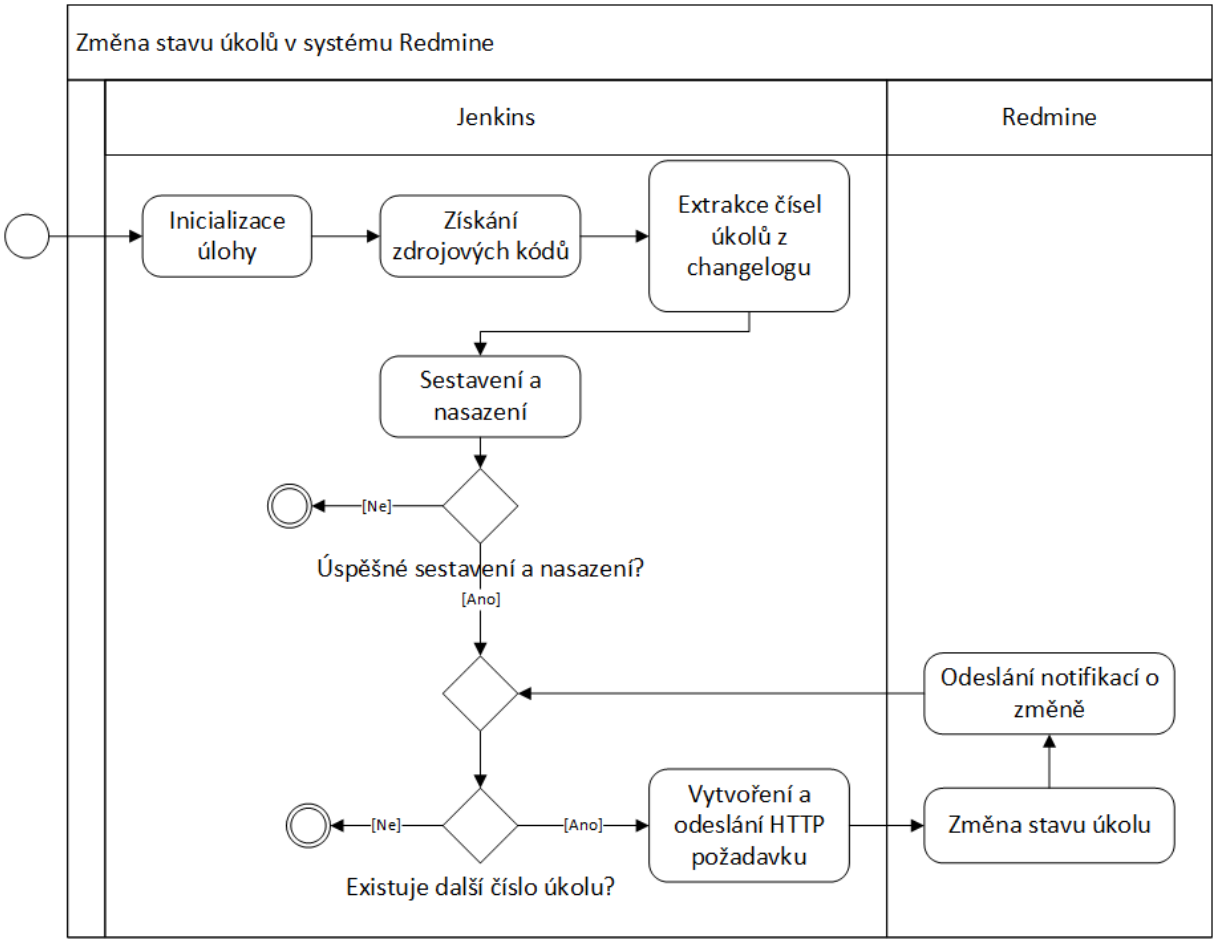
První realizaci určenou pro otestování a validaci myšlenky, jsem udělal přímo v úkolu pro sestavení a to pomocí skriptů. Následně by bylo možné tyto skripty zaintegrovat do existujícího rozšíření nebo vytvořit rozšíření nové.

Jenkins získává změny ve zdrojových kódech rozdílem mezi aktuálním spuštěním úlohy a předešlým spuštěním. Proto jsem se rozhodl, že skript na změnu stavu umístím do úlohy, která se starala o sestavení všech modulů. Toto byl i první impulz k tomu, že by aplikace pohodlne.info měla být brána jako celek a bude stačit jedna úloha, která by se postarala o všechny moduly.

Změna stavu se skládá ze dvou částí. Celé schéma změny stavu je přehledně zobrazeno na obrázku 12. V první částí se získají identifikátory všech změněných úkolů ze změn (changesetu) poskytnutých Gitem (na obrázku jako „Extrakce čísel z changelogu“). Pro potřeby sdílení skriptu mezi více úlohami jsem použil rozšíření Managed Scripts, které spravuje globální skripty a je použitelné ve všech úlohách. Skript je napsaný v Groovy a je k dispozici v příloze B jako zdrojový kód číslo 4.

Druhou částí bylo po úspěšném dokončení sestavení a nasazení provést samotnou změnu stavu. To jsem pro jednoduchost realizoval v Bash skriptu, který je opět uložen globálně a v konkrétní úloze je spuštěn. Skript pomocí linuxového programu Curl volá Redmine API, tak aby došlo ke změně stavu konkrétního úkolu (na obrázku 12 jako „Vytvoření a odeslání HTTP

požadavku“). Jako parametry slouží vždy číslo úkolu, stav, do kterého se má úkol přepnout, a poznámka. Skript je opět k dispozici v příloze B jako zdrojový kód číslo 5.



Obrázek 12: Schéma komunikace mezi systémem Jenkins a Redmine pro změnu stavu úkolu

Použitá rozšíření

Rozšířeních, která jsem postupně instaloval a konfiguroval, bylo několik, některá se zapojila přímo do jednotlivých úloh, některá sloužila jen pro pohodlnější konfiguraci a používání systému Jenkins.

Git plugin – získávání zdrojových souborů ze systému Git, ověřování prostřednictvím SSH klíče.

Extra Columns Plugin – přidává sloupce do seznamu úloh.

Managed Scripts – globální úložiště pro skripty, použitelné ve všech úlohách.

Multijob plugin – umožní vytvořit úlohu skládající se z více úloh.

Role-based Authorization Strategy – rozšíření pro definici oprávnění založených na rolích a jejich přiřazení uživatelům.

Publish over SSH – rozšíření umožňující odeslat soubory prostřednictvím SCP protokolu.
ThinBackup – rozšíření pro automatickou zálohu celého systému Jenkins.

Druhá verze úloh Nějakou dobu jsem praktickým spouštěním testoval první verzi úloh (a to jak případy kompletního, tak i jednotlivého sestavení a nasazení). Problém nastal při potřebě změnit nějaké nastavení jednotně u všech úloh (např. větev, ze které se má sestavení provádět, rozšíření nasazení na další server). To se ukázalo jako velmi složité, zdlouhavé a náchylné k chybě.

Proto jsem začal implementovat druhé řešení, ve kterém byly společné části extrahovány do jedné úlohy. Dále jsem vytvořil úlohy, které s přednastavenými parametry volaly hlavní výkonnou úlohu. Problém nastal při větším počtu běžících úloh. Protože Jenkins obsahuje nastavitelný parametr stanovující počet souběžně běžících úloh a tento parametr byl při spuštění všech úloh mnohonásobně překročen. To ve výsledku znamenalo, že se spustily pouze malé úlohy a na volání té výkonné už došly zdroje. Určitým řešením by bylo navýšení počtu souběžně běžících úloh, to už ale nebylo z důvodu zátěže serveru možné.

Třetí verze úloh Poslední a aktuálně využívaná verze vyšla z pozorování, že samostatné sestavení a nasazení jednotlivých modulů není potřebné, viz obrázek 8, a plně dostačuje sestavit a nasadit všechny moduly naráz. Podstatný vliv mělo zjištění, že aplikační server Tomcat společně s Liferay aplikací nezvládá spolehlivě nasazení větších aplikací za běhu. Což je způsobeno problémy s ClassLoaderem a neodstraňováním starých souborů.

V repozitáři s projektem jsem vytvořil Maven skript, který postupně zavolal všechny ostatní skripty modulů a jejich výsledky shromáždil v jedné složce. K tomu jsem využil rozšíření Maven-dependency-plugin. To je jediný skript, který se z prostředí Jenkins volá. Další výhodou tohoto řešení je možnost si stejné sestavení spustit i lokálně pro rychlý test všech modulů.

Nastavil jsem automatickou kontrolu repozitářů (úloha se spustí, pokud dojde ke změně), která se vykonává periodicky dvakrát za den (v době, kdy jsou servery nejméně vytížené). Vzhledem k počtu změn je tato doba dostačující a výhody jako včasná zpětná vazba jsou zachovány. Současně není pro testery problém počkat na druhý den, kdy k automatickému spuštění dojde. V případě nutnosti stále existuje možnost spustit sestavení ručně kdykoliv.

Posledním krokem nastavení automatického sestavení jsem se přiblížil k CI (podle Martina Fowlera [15] je pravidelné spouštění sestavení jednou za den krok, který k CI vede, ale o CI se v pravém slova smyslu nejedná). Nicméně sestavení po každé změně by se dalo snadno dosáhnout s rozšířením výpočetních prostředků.

Další aplikace Postupně jsem pracoval na konfiguraci úloh pro další vyvíjené programy. Pro EditorŠVP existoval Ant skript pro sestavení, skript však nesestavil kompletní nasaditelnou aplikaci, pouze její část. Proto jsem Ant skript doplnil tak, aby produkoval nasaditelný `.war` soubor.

Automatické nasazování

Automatické nasazování jde realizovat několik způsoby. První způsob je vyvolání akce ze strany Jenkins serveru a odeslání souboru prostřednictvím SCP. Druhou variantou je nahrát sestavený artefakt na určené místo, odkud si jej daný server sám získá a aktualizuje svoje balíčky. První variantu jsem zvolil z důvodu jednoduchosti a univerzálnosti. Na straně serveru nemusí běžet žádný speciální skript nebo software, který by zařizoval aktualizace a jde použít pro většinu serverových aplikací. Toto řešení má i blíže k populárnímu PaaS[13], kdy se pouze poskytne balíček s aplikací a o provoz middleware a serveru se postará poskytovatel služby.

Poslední řešenou otázkou je, zda má být nasazování na produkční server součástí jednoho úkolu jen jako poslední krok (automaticky nebo čekající na potvrzení), anebo je lepší použít variantu, kdy jedna úloha řeší sestavení a další nasazení s výběrem prostředí. První varianta je bližší k CD, ale je potřeba kvalitní pokrytí testy. Vzhledem k malému počtu prostředí (pouze testovací a produkční) a stylu vývoje, je zvolená varianta s výběrem prostředí vhodná. Nicméně, pokud by se v dalším vývoji podařilo posunout blíže k CD a častějšímu nasazení verze do produkčního prostředí, bylo by vhodné uvažovat o jedné úloze.

Budoucnost

V budoucnu bych chtěl aktualizovat Jenkins na verzi >2 s podporou „pipeline“, kdy se úlohy definují jako kód, který je součástí repozitáře. Toto řešení přináší lepší ovladatelnost a snadnou historizaci celé úlohy.

5.5 Testování

V oblasti testování jsem firmě pomohl v několika ohledech. První kontinuálně testovanou položkou je sestavení aplikace. S využitím nástroje Jenkins se aplikace začaly v pravidelných intervalech sestavovat automaticky a při zavlečené chybě, která by způsobila nekompilovatelnost, je chyba včas odhalena.

Nasazením softwaru Jenkins se také připravil prostor pro zapojení automatického testování kvality kódu např. prostřednictvím nástroje SonarQube, který bude v nejbližší době nainstalován a nakonfigurován tak, aby se chyby, které jsou zjistitelné statickou analýzou, našly co nejdříve a nejrychleji.[14]

5.5.1 Systémové testy pomocí nástroje Selenium

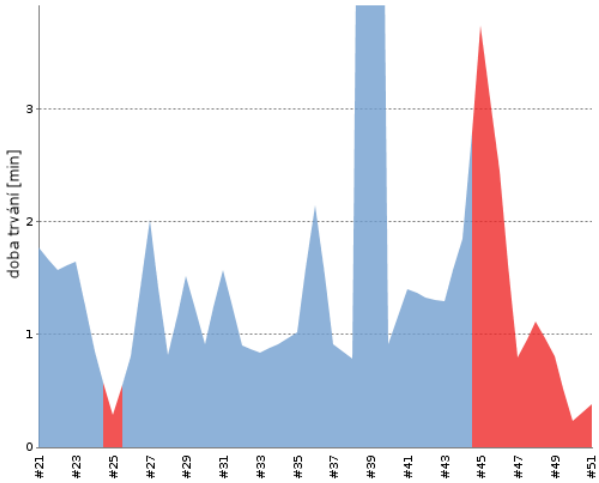
Pro test ověření základních funkčností jsem zvolil nástroj Selenium. Selenium je nástroj pro automatické testování webových aplikací, simuluje práci uživatele v prohlížeči a lze ho spouštět i z prostředí jazyka Java, například v rámci životní cyklu Maven.

[13] PaaS - Platform as a Service - platforma jako služba

[14] V rámci semestrálního projektu jsem propojení mezi nástrojem Jenkins a SonarQube úspěšně realizoval.

Vytvořil jsem základní test, který načte dané stránky portálu pohodlne.info a ověří, zda při vykreslování nenastala chyba (hledá v HTML element s příslušným ID, které označuje chybu). Tento test lze automaticky spouštět po novém nasazení aplikace v bezoknovém prohlížeče Phantom.JS. Vzhledem k tomu, že se jedná o systémový test, je potřeba kompletní aplikace včetně testovacích dat. Na druhou stranu je test velmi rychlý a lze jej použít jako základní smoke test.

Na obrázku 13 jsou zobrazeny záznamy provádění základního testu, spouštěného nástroji Jenkins. V grafu jsou zobrazeny jednotlivé provedené testy a doba jejich trvání v minutách (závisí na aktuálním stavu serveru, který poskytuje odpovědi). Červeně označené běhy znázorňují test s chybou. Modré jsou testy, které úspěšně proběhly.



Obrázek 13: Záznam z vykonávaných základních testů z nástroje Jenkins

Dalším vytvořeným systémovým testem byl test, který po hromadném importu skrz RSS získá a projde všechny URL adresy a stejně jako v předchozím případě kontroluje výskyt chyb.

5.5.2 Testování konfigurace

Díky použití technologií Vagrant a Puppet (viz kapitola 5.6.1) může docházet i k neustálému testování konfigurace. Kdy se periodicky může vytvořit nový server, aplikovat na něj konfigurace a ověřit funkčnosti. Nejedná se jen o funkční testování, ale neustálým opakováním dochází i ke kontrole míry automatizace.

5.6 Provoz a správa serverů

Další velkou kapitolou, kterou jsem se zabýval, je provoz a správa serverů. Ve firmě není žádné specializované oddělení ani člověk, který by se touto prací zabýval. Úkoly spojené s provozem má vždy na starost jeden z vývojářů. Všichni vývojáři jsou schopni základní práce s linuxovým systémem, ale většina vývojářů pracuje v operačním systému Windows. Pro provoz virtuálního

operačního systému v rámci OS Windows existuje několik programů a mezi ty nejpopulárnější v oblasti jednoduché virtualizace patří Oracle VM VirtualBox.

5.6.1 Vagrant

Vagrant je open-source nástroj, který výrazně usnadňuje práci s virtuálními stroji. Je to rozhraní, pod kterým může pracovat VirtualBox, VMWare nebo Hyper-V. Stejně jednoduše je možné spouštět i kontejnery Docker. Oproti přímému používání VirtualBoxu umožňuje například po startu stroje automaticky spouštět autokonfiguraci serverů (Puppet, Salt, apod.) nebo jednoduché a bezproblémové sdílení složek. Konfigurace se zapisuje do souboru, lze ji tedy verzovat a snadno přenášet [28].

Operační systém se do prostředí nástroje Vagrant dostává prostřednictví balíčku (boxu). Na stránce `https://atlas.hashicorp.com/boxes/search` je k výběru velké množství základních boxů. Lze vytvořit i box vlastní, který bude mít základ z existujícího a bude obsahovat navíc libovolnou konfiguraci.

Spojení těchto vlastností přináší možnost jedním příkazem vytvořit lokální prostředí stejné se serverovým.

Simulace testovacího a produkčního prostředí byla ve firmě velkým problémem, který jsem pomocí nástroje Vagrant vyřešil. V příloze C je ukázka souboru, který definuje dva virtuální stroje a jejich konfiguraci. Po spuštění příkazu `vagrant up` se vytvoří dva virtuální stroje s danými parametry a spustí se autokonfigurace prostřednictvím nástroje Puppet.

Takto vytvořené virtuální stroje lze jedním příkazem absolutně vyčistit a dalším inicializovat do výchozího stavu. Je to naprosto ideální způsob pro testování konfigurace.

Rychlé a jednoduché zkoušení aplikací je pomocí Vagrantu v kombinaci s Dockerem snadné i na OS Windows. Vagrant poskytne Linuxové prostředí a Docker kontejner běžící aplikaci. O předávání portů mezi hostujícím a hostovaným počítačem se postará Vagrant.

Dalším problémem ve firmě byla neschopnost replikace serverového prostředí, ať už pro potřeby migrace, rozšíření nebo testování. Konfiguraci spravovali různí lidé a dokumentace nebyla aktuální. Firma pro testovací a vývojové účely využívá 3 fyzické servery a jeden server v cloudu, který slouží jako produkční prostředí pro aplikaci pohodlne.info. Dále firma spravuje aplikace ISIS na několika zákaznických serverech (zde se úroveň konfigurace liší podle smluv). A v neposlední řadě pracuje se servery pro aplikaci EVSRŠ (na úrovni aplikace). V případě aplikace SMILE firma pomáhá zákazníkům s jejich vlastní instalací. I přesto, že počet spravovaných serverů není příliš velký, se začaly objevovat problémy. Pro budoucí rozvoj firmy je možnost snadné správy více serverů velmi důležitá.

Existuje několik nástrojů a technik, které přináší řešení těchto problémů. Mezi hlavní techniky patří využití IaC (viz kapitola 3.6 Automatizace infrastruktury, infrastruktura jako kód),

které zaručí živou a znovu aplikovatelnou konfiguraci s velkou mírou sebedokumentace. Existuje několik nástrojů, které s tímto konceptem pracují, mezi nejznámější patří Puppet, Chef, Salt nebo Ansible. Princip všech těchto nástrojů je stejný – automaticky aplikují na dané servery danou konfiguraci. Liší se přístupem, stylem zápisu konfigurace a rozsahem použití. Například nástroj Ansible funguje prostřednictvím SSH protokolu (nepotřebuje speciální aplikaci na konfigurovaném server).

5.7 Docker

Docker je technologie, která umožňuje spouštět kontejnery. Což jsou zjednodušeně řečeno rychlé a hardwarově nenáročné virtuální stroje. Na běžném počítači může v jednu chvíli běžet i více než 20 Docker kontejnerů. Prostor běžícího kontejneru je striktně oddělen od hostitelského počítače a veškeré změny, které v rámci běhu kontejneru vzniknou, se po vypnutí kontejneru vymažou. Pro persistenci dat se využívá propojení a namapování složek hostitelského počítače. Obsah v takových složkách po vypnutí kontejneru zůstane nezměněný.

Toto řešení přináší celou řadu výhod. Od jednoduché správy, přes snadný přenos na jiný stroj, až po možnost rychlého testování v nezávislém prostředí.

Docker lze dále využít například jako prostor pro běh sestavení. Díky spuštění v takovém kontejneru bude sice proces sestavení trvat ve většině případu déle než sestavení na nativním hardware, ale je zajištěna opakovatelnost. Pokud je kontejner nastaven tak, aby fungoval, bude beze změny funkční i při použití na jiném stroji.

Nevýhodou použití můžou být mírně vyšší hardwarové nároky než při klasickém nativním běhu. Hodně ale záleží na povaze aplikace, pokud se jedná o databázový server, který ve velké míře využívá zápis a čtení z disku (obecně persistenci), může být výkon kontejneru mnohem nižší v porovnání s nativní aplikací. Pokud je aplikace roztříštěna do velkého množství kontejnerů může být složitější jejich propojení, synchronizace a komunikace.

Nástroj Docker jsem využil pro instalaci prostředí GrayLog. Testoval jsme v něm také aplikaci RocketChat na komunikaci a vyšší verzi nástroje Jenkins.

5.8 Puppet - nástroj pro autokonfiguraci systémů

Puppet je nástroj založený na jazyce Ruby, který slouží k rychlému, spolehlivému a opakovatelně použitelnému spouštění autokonfigurace serverů bežících pod OS Linux, UNIX i Windows. Lze ho použít v celém životním cyklu prostředí, od jeho vzniku, užívání, aktualizace, až po ukončení a rušení (případný přesun) [29]. Dokáže fungovat ve dvou režimech:

1. Client-server – existuje jeden hlavní server – master, který poskytuje konfiguraci jednotlivým uzlům – klientům,
2. Samostatně – provoz bez serveru, kdy se konfigurace doručuje na jednotlivé servery jiným způsobem a je na nich přímo spouštěna.

Puppet kromě své bezplatné verze nabízí i verzi Enterprise, ke které je dostupná podpora a další rozšíření. Nicméně během používání bezplatné varianty jsem nenarazil na chybějící funkce.

Puppet sám o sobě poskytuje pouze základní funkčnost a jeho síla je v poskytovaných modulech. Na `https://forge.puppet.com/` je k dispozici více než čtyři tisíce modulů, které obalují práci s různými programy. Moduly se instalují na master serveru a to jednoduchým příkazem. Také lze použít rozšíření Librarian-puppet a definovat moduly prostřednictvím `Modulefile`. Z počátku jsem pro jednoduchost využil první možnost.

V první fázi jsem Puppet použil v módu client-server pro konfiguraci produkčního prostředí na serveru poskytovaném Google Cloud Engine. Na tento server jsem v rámci „startup scriptu“[15] nainstaloval Puppet klienta a nastavil ho tak, aby získával konfiguraci od master serveru.

Po nastavení serveru a otevření příslušných portů (TCP:8140) se klient přihlásí na server a po potvrzení začne ve stanovém čase probíhat výměna informací (katalogů) o konfiguraci. Od té doby jsem měnil konfiguraci v souborech na serveru a klient se vždy po určité době nakonfiguroval podle zadání. Pro testování konfigurace jsem používal kombinaci s nástrojem Vagrant, což umožnilo testovat produkční konfiguraci na lokálním stroji. Veškerý kód se kterým Puppet pracuje je verzován do samostatného Git repozitáře.

5.8.1 Provedená konfigurace na produkčním serveru

Server pro běh produkční aplikace od Google Cloud Engine vychází z čisté instalace OS Debain 8. Pomocí Puppet jsem doinstaloval Oracle Javu, MySQL databázi a nainstaloval a nakonfiguroval Apache Tomcat jako aplikační server a Nginx jako web server, který slouží jako proxy a předává vnitřně požadavky na Tomcat. Ukázka takové konfigurace je v příloze D.

Dalším krokem byla instalace aplikace Liferay. Liferay je distribuovaný jako `.war` soubor a několik dalších externích knihoven. Dále je potřeba dodat konfiguraci pro připojení do databáze a v neposlední řadě vytvořit adresářovou strukturu pro data a index, které Liferay za běhu používá.

5.8.2 Liferay modul

Na řádku 99 v příloze D je použití vlastního vyvinutého modulu pro instalaci a konfiguraci Liferay. Jako parametry vstupují cesta k instalaci serveru Tomcat, požadované číslo Liferay verze, potřebné údaje k připojení k databázi a požadovaná cesta k Liferay souborům.

5.8.3 Konfigurace na vývojovém serveru

Puppet ovlivňuje pouze to, co je v konfiguraci zapsané, takže ho lze využít i na stávajícím manuálně nakonfigurovaném serveru, postupně přepisovat konfiguraci a nové věci už přidávat prostřednictvím nástroje Puppet.

[15] `https://cloud.google.com/compute/docs/startupscript`

Této vlastnosti jsem využil při instalaci nových aplikací na vývojový server, takže veškeré nové instalace a konfigurace jsou vytvořeny právě přes Puppet a staré se budou příležitostně přepisovat. Bohužel, kvůli rozdílným verzím OS Linux na tomto vývojovém serveru a serveru poskytujícím konfigurace, jsem nedokázal sladit verze programu Puppet, a proto je nutné spouštět Puppet v samostatném režimu. Což v praxi znamená získání změn z Git repozitáře a aplikaci těchto změn prostřednictvím příkazu `puppet apply`.

O dalších příkladech konfigurace na vývojovém serveru je napsáno v kapitole 6.3 Rozdělení testovacího aplikačního serveru. Puppet jsem úspěšně využil i při přípravě konfigurace z další kapitoly.

5.9 Sledování a monitoring

Poslední větší částí, se kterou jsem pracoval, bylo sledování a monitoring. Pod tuto oblast spadá jak získávání informacích o běhu aplikací, tak informace o chování a běhu systému. Většina aplikací ve firmě zapisuje informace z běhu prostřednictvím Log4j. Pouze jediná instalace aplikace ISIS nějakým způsobem dále s těmito daty pracuje. Informace o chybách jsou odesílány prostřednictvím emailu. Nicméně tyto emaily většinou končí neanalyzovány a je velice složité podle nich problémy nějakým způsobem řešit.

Informace zaznamenané při běhu ostatních aplikací se automaticky neanalyzují a nijak se s nimi dále nepracuje. Informace se z nich získávají pouze v případě potřeby, vždy ad-hoc na konkrétním serveru, kde běží daná aplikace. V nepravidelných intervalech (nejčastější při restartu serveru) dochází k vymazání starších logů.

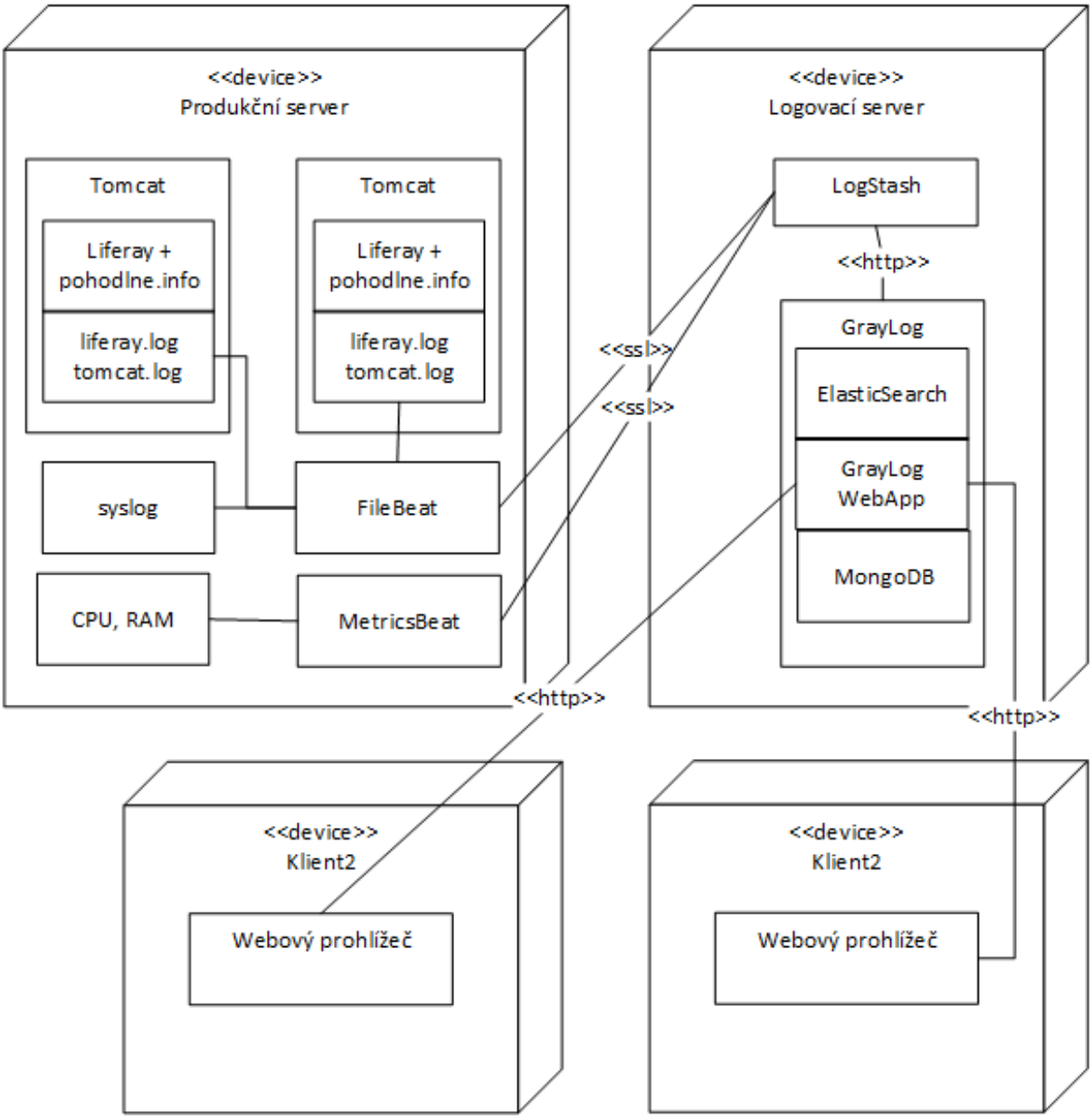
Pro snadnou práci s daty z běhu aplikace existuje technika centrálního logování (viz kapitola 3.8 Nepřetržité a centrální monitorování).

Pro implementaci centrálního logování jsem zvolil použití programů z rodiny Beats na monitorovaných serverech[16].

Na centrálním serveru program Logstash data z Beats přijme a zpracuje. Zde by se v případě potřeby mohla vložit fronta, který by sloužila jako mezipaměť v případě větší zátěže. Program Logstash data zpracuje a připraví. Výstup z programu Logstash směřuje na vstup aplikace GrayLog. Aplikace GrayLog data uloží do ElasticSearch databáze a zpřístupní je přes webové rozhraní. Schéma celé komunikace je znázorněno na obrázku 14.

První částí, ještě před konfigurací Beats pro odesílání logu, je ale samotná příprava zdrojových dat, které jsou pak následně odeslány na centrální server, kde jsou nad nimi prováděny další akce.

[16] Existuje i několik dalších variant pro přenos logů, například lze Logstash napojit přímo na Log4J. Já jsem se rozhodl pro FileBeat z důvodu jeho univerzálnosti.



Obrázek 14: Schéma přenosu dat z produkčního serveru na server pro centrální logování

5.9.1 Data z aplikace pohodlne.info

Prvním úkolem bylo získávat logy z aplikace pohodlne.info. Aplikace pohodlne.info je serverová Java aplikace využívající portál Liferay. Liferay spuštěný na aplikačním serveru Tomcat ve své výchozí instalaci všechny monitorovací zprávy vypisuje v několika různých formátech do několika souborů:

- `liferay/logs/liferay.yyyy-mm-dd.log` standardní výstup Log4J, konfigurovaný pomocí `portal-log4j.xml`,

- `tomcat/logs/catalina.out` zaznamenává `stdout` a `stderr` proud, přesměrovaný do souboru ve spouštěcím skriptu `catalina.sh`,
- `tomcat/logs/catalina.yyyy-mm-dd.log` standardní log serveru Tomcat, konfigurovatelný pomocí `tomcat/conf/logging.properties`,
- `tomcat/logs/localhost_access_log.yyyy-mm-dd.txt` standardní záznamy o přístupu na server, konfigurovatelný pomocí `tomcat/conf/server.xml`,
- `tomcat/logs/host-manager.2016-01-16.log` a `tomcat/logs/manager.2016-01-16.log` soubory pro záznam informací z aplikací manager a host-manager, které jsou součástí výchozí instalace Tomcatu, nakonfigurované v souboru `logging.properties`,
- `tomcat/logs/tomcat.log` informace vypisované serverem Tomcat, konfigurovatelné prostřednictvím `logging.properties`.

V několik souborech se objevují stejné informace a formát napříč soubory je odlišný a neobsahuje všechna potřebné data pro pohodlné zpracování a využití v centrálním serveru pro správu monitorovacích záznamů. Některé soubory jsou nevyužitelné.

Postup úprav a optimalizace

Po výše zmíněné analýze následovaly následující úpravy v několika konfiguračních skriptech [30]:

- `logging.properties` v tomto souboru jsem upravil nastavení tak, aby se nevytvářely nevyužívané soubory pro aplikace manager a web-manager.
- `log4j.properties` úprava formátu data, odstranění redundantních nastavení, které způsobovaly vícenásobné logy do souboru `tomcat.log`.
- `portal-log4j-ext.xml` tento soubor slouží pro úpravu výchozího způsobu logování aplikace Liferay. Nastavení v tomto souboru přepisuje výchozí nastavení logování. Upravený soubor se umisťuje do složky `liferay/WEB-INF/classes/META-INF` společně s popisným `log4j.dtd`. V tomto souboru jsem nastavil formát pro datum jako v `log4j.properties`. Ukázka nastavení ze souboru `portal-log4j-ext.xml` je ve výpisu číslo 1.

```
<appender name="FILE" class="org.apache.log4j.rolling.RollingFileAppender">
   <rollingPolicy class="org.apache.log4j.rolling.TimeBasedRollingPolicy">
      <param name="FileNamePattern" value="@liferay.home@/logs/liferay@spi.id@
         .%d{yyyy-MM-dd}.log" />
   </rollingPolicy>
   <layout class="org.apache.log4j.EnhancedPatternLayout">
      <param name="ConversionPattern" value="[%d{ISO8601}]: [%p]-[%m]%n" />
   </layout>
</appender>
```

Výpis 1: Část konfiguračního souboru `log4j-ext.xml`

Některé logy se v aplikaci Liferay zapisují přímo na standardní výstup – změna tohoto chování by znamenala rekompilaci celého základu aplikace Liferay. Některé zprávy tak zůstávají v souboru `catalina.out`, tyto zprávy ale nejsou důležité a nemá smysl se s nimi dále zabývat.

Soubory, které budou postupovat k dalšímu zpracováním jsou `liferay.yyyy-mm-dd.log` a `tomcat.log`.

Starší nastavení aplikace Evidence

Aplikace Evidence využívá nastavení `log4j.properites`, které je umístěné v globální složce aplikačního serveru, toto nastavení je už poměrně dlouho neudržované a v současné chvíli způsobuje komplikace například několikanásobný výpis informací do souboru `tomcat.log`. Analýzou tohoto nastavení jsem objevil příčinu v nastavení úrovně pro několik různých balíčků, které ale všechny patřili pod `org.apache`. Tento soubor jsem tedy upravil tak, aby logy vypisoval neduplicitně a v jednotném formátu. A následně jsem upravil sestavovací skript tak, aby se soubor s nastavením pro logování vkládal přímo do produkovaného `.war` souboru.

Data jsou nyní připravena v co možná nejjednotnějším formátu pro přenos na centrální server prostřednictvím programu z rodiny Beats.

5.9.2 Transport dat prostřednictvím Beats

Platforma Beats[17] je vyvinutá společností Elasticsearch jako lehké řešení pro přenos dat do programu Logstash nebo přímo do Elasticsearch. Existuje několik programů, které jsou specializované na přenos různých dat. Zprávy jsou při přenosu zabezpečeny proti odposlechu a ověření autentičnosti odesilatele pomocí SSL. Pro řešení aktuálního problému jsem využil dva programy[18]:

FileBeat – transportuje logy ze souborů, pamatuje si poslední úspěšně odeslané řádky, umí komunikovat s příjemcem a upravovat parametry přenosu (například zpomalit přenos),

MetriscsBeat (do verze 5.0.0 TopBeat) – zaměřen na získání a transport číselných dat o běhu (ať už operačního systému nebo jiné aplikace).

Pro distribuci konfigurací programů FileBeat a MetricsBeat jsem využil Puppet, do kterého jsem doinstaloval modul pro snadnější práci právě s těmito programy. Základní konfigurace programu FileBeat (viz výpis 2) obsahuje cestu ke kontrolovaným souborům, přidané pole pro identifikaci zdrojové aplikace a nastavení pro identifikaci víceřádkových logů.

[17] `https://www.elastic.co/products/beats`

[18] Dalšími jsou Packetbeat - síťové informace, Winlogbeat - události OS Windows, Heartbeat - informace o dostupnosti dané aplikace.

```
filebeat::prospector { 'tomcat7-digis-logs':
       paths    => [
         '/opt/tomcat7-digis/logs/*.log',
         '/opt/liferay-digis/logs/*.log',
       ],

       fields_under_root => true,
       fields => { "application_name" => "liferay-digis.vrk.cz" },


        multiline => {
          pattern => '^\[[0-9]{4}-[0-9]{2}-[0-9]{2}',
          negate => 'true',
          match => 'after'
       }
   }
}
```

Výpis 2: Ukázka konfigurace programu FileBeat v Puppet skriptu pro odesílání logu ze souborů generovaných běžící aplikací

MetricsBeat jsem nakonfiguroval pro přenos informací o zátěži procesoru a spotřebě RAM paměti.

Díky použití nástroje Puppet a centrální správě serverů jsem získal možnost snadno a spolehlivě tyto nastavení měnit z jednoho místa. Nevzniká tak potřeba přenášet manuálně konfigurační soubory mezi servery a v případě dalšího rozšíření bude vše jednoduše udržitelné.

Příjem a zpracování dat prostřednictvím programu Logstash

Logstash slouží pro zpracování dat z logů a přijímá data z programu FileBeat a MetricsBeat. Nainstaloval jsem ho na vývojovém serveru firmy a upravil konfiguraci sítě tak, ať je dostupný i z internetu. Po přijetí dat následuje sekvence příkazů, po kterých se logy dostanou do podoby, se kterou se dobře pracuje v programu GrayLog.

Konfigurace programu Logstash se skládá ze tří částí:

- `input` – definice vstupů, které poskytují data pro zpracování,
- `filter` – zpracování dat,
- `output` – definice výstupů pro odeslání zpracovaných dat.

Nejzajímavější je část `filter`, kterou jsem nakonfiguroval tak, že se do logů, které v sobě neměly žádnou zprávu (data z MetricsBeat obsahují informace v jiných polích než `message`),

což způsobovalo nekorektní chování programu GrayLog, přidává zpráva. Další část zpracování je zaměřená na zprávy s více řádky (typicky StackTrace po vyvolání výjimky). Pomocí rozšíření Grok[19] se text pole `message` kontroluje na obsah `(?m)\n%{GREEDYDATA:additional_info}` a pokud obsah vyhovuje tomuto výrazu, extrahuje se do pole `additional_info`.

Na výstupu se data odešlou na URL adresu pro příjem programu GrayLog.

Prezentace a práce s daty v aplikaci Graylog

GrayLog je komplexní open-source systém pro ukládání, zobrazování a analýzu logovacích dat. Je snadno škálovatelný (logy lze zpracovávat na více uzlech). Na rozdíl od populárního nástroje Kibana, který také slouží pro zobrazování logovacích dat a grafů, obsahuje GrayLog možnosti pro práci s uživateli a další nastavení. Umožňuje vytvoření pravidel pro zpracování zpráv a obsahuje i systém na upozorňování. Pro svůj provoz využívá ElasticSearch pro ukládání zpracovaných dat, MongoDB pro ukládání konfigurace a samotný GrayLog, který se skládá z programu pro zpracování dat (Java) a webového rozhraní (Node.JS).

Vzhledem k tomu, že jsem tento program v první fázi testoval a nebyl jsem přesvědčen o finálním použití a umístění na konkrétní server, zvolil jsem instalaci prostřednictvím kontejnerové technologie Docker. Celé prostředí pro provoz programu GrayLog tak závisí na jediném souboru s definicí kontejnerů a minimální přidané konfiguraci (nastavení souborových cest apod.). Tento soubor je součást v přílohy E.

Další nastavení, například pro správné odesílaní notifikačních emailů, jsem provedl v souboru `graylog.conf`. Mimo jiné lze v tomto souboru nastavit i jak bude GrayLog dělit data do ElasticSearch indexů, podle jaké strategie bude staré indexy zavírat (respektive mazat) a kolik jich má uchovávat. Pro rotování indexů jsem zvolil časovou strategii, takže jeden index bude otevřený dva týdny. Při reálném použití se do jednoho indexu uložilo cca 450 000 zpráv. Na disku zabírá jeden index cca 380 MB. Dále jsem zvolil, že maximální počet uložených indexů bude 6. Při tomto nastavení budou přístupny informace tři měsíce zpátky, což je v tuto chvíli dostačující doba. Ukázka tohoto nastavení je ve výpise číslo 3.

```
rotation_strategy = time
elasticsearch_max_time_per_index = 2w
elasticsearch_max_number_of_indices = 6
```

Výpis 3: soubor `graylog.conf` – část nastavení programu GrayLog pro zachování indexů ElasticSearch

[19] program pro rychlé a snadné parsování textových dat

Práce s daty ve webovém rozhraní Graylog

Webové rozhraní nabízí základní funkce jako správu uživatelů a jejich práv, statistické informace o využití, nastavení vstupů a výstupů. Dále Graylog umožňuje vytvořit další úroveň zpracování dat po přijetí. Jeho hlavní funkcí je ale zobrazování dat a výstupů.

Je možné pokládat ad-hoc dotazy s aplikovaným filtrem podle jednotlivých polí, dotazy lze omezit podle data nebo jiných konkrétních hodnot. Výsledkem dotazu je přehledný seznam všech odpovídajících záznamů. Na záznamy lze získat jedinečný a neměnný odkaz (ideální pro sdílení napříč týmem). Současně lze z nalezené zprávy přejít na zprávy v podobném čase a hledat tak mezi daty souvislosti.

Pro rozdělení nebo jako možnost omezení uživatelů existují streamy, které jednotlivé zprávy podle pravidel agregují. Po přidání dalších zdrojů logů a informací jsem vytvořil několik takových streamů.

Grafy a použití dashboardů Graylog dokáže na základě logů vykreslovat grafy nebo statistiky četnosti, které se seskupují do dashboardů. Lze tak jedním pohledem zkontrolovat stavy systému nebo aplikace. Pro rychlý přehled o zdrojích na jednotlivých serverech jsem vytvořil dashboard „Systems“ – obrázek 15. Grafy mohou zobrazovat počet logů s úrovní ERROR v čase.

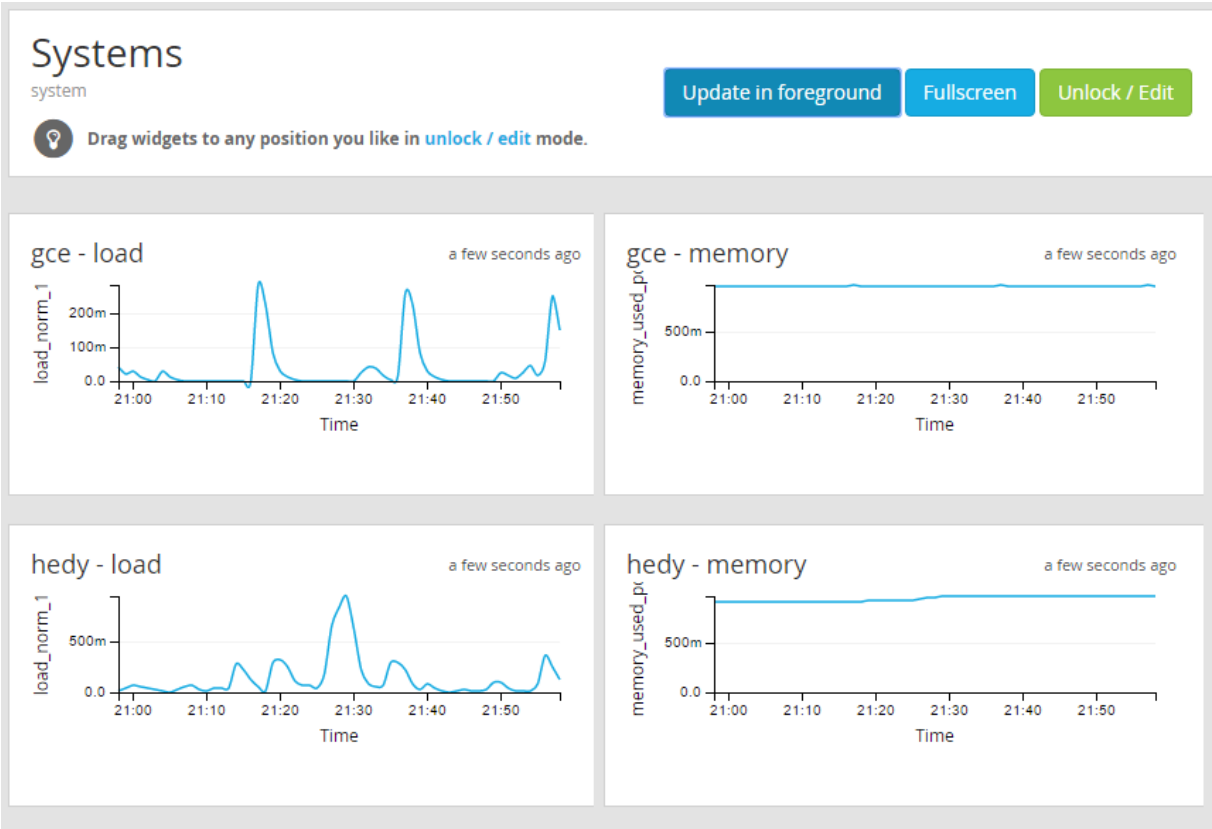
5.9.3 Další monitorované aplikace a systémy

Následně jsem rozšířil získávání logů o další aplikace a testovací prostředí pro snadný přístup vývojářů k informacím.

Možné rozšíření, budoucnost Postupně budu do systému monitorování zapojovat další aplikace a servery. Mezi konkrétní kroky patří i napojení na systém Redmine pro automatické vytváření úkolu při výskytu logu úrovně ERROR v produkčním systému. Další možnou integrací je napojení na systém Mandrill, který slouží na hromadné odesílání emailů. Tento systém dokáže prostřednictvím HTTP požadavků produkovat zprávy o použití (počty odeslaných, neodeslaných emailů apod.). Mandrill sice tyto informace zobrazuje přehledně ve svém rozhraní, ale získání alespoň přehledových informací přímo do programu GrayLog by zjednodušilo každodenní kontroly a správu.

5.10 Shrnutí aplikovaných nástrojů

V této kapitole jsem popsal řešení konkrétních úkolů, tabulka 1 přehledně zobrazuje výše popsané nástroje a technologie společně s hlavními přínosy jejich využití oproti použití původních nástrojů a postupům.



Obrázek 15: Dashboard zatížení procesoru a spotřeba RAM paměti v procentech.

Oblast	Původní nástroj	Aktuální nástroj	Největší změny
Plánování	Redmine	Redmine	automatická aktualizace stavů
Sdílení zdrojových kódů	Git	Git	přehlednější commity, popisnější zprávy, používání s ohledem na CI
Sestavení	Maven, (Ant)	Maven	optimalizování skriptů s ohledem na CI
Spouštění sestavení	Shell skripty	Jenkins	přehlednější, udržitelné, snadno rozšiřitelné, častější spuštění, cesta k CI
Nasazení	Manuální kopírování	Jenkins	omezení manuálních chyb - spolehlivější, rychlejší a častější provádění
Konfigurace prostředí	Manuálně, ad-hoc	Puppet, Docker	omezení manuálních chyb, možnost znovupoužití konfigurace, snadná distribuce změn, živá dokumentace, snadná instalace a migrace podpůrných aplikací
Testování konfigurace	-	Puppet, Vagrant	snadné získání totožného prostředí, možnost testování změn
Čtení logů a monitoring	Manuální SSH přihlášení a hledání v souborech	FileBeat, Logstash, Graylog	snadnější přístup k informacím, možnost rozšířeného vyhledávání, navázání dalších akcí, agregace dat

Tabulka 1: Přehled původních a nově používaných technologií a přehled hlavních přínosů používání nových nástrojů

6 Analýza dopadu změn na vývojový proces

Poslední část této diplomové práce popisuje, jaký dopad měly jednotlivé změny, použité nástroje a praktiky na vývojový proces ve firmě VRK plus s.r.o. První část se zabývá připomenutím problémů, které by s novými nástroji a postupy nenastaly. Další část se zaměřuje na výraznou změnu v nasazování. Následující část rozebírá řešení úkolů pomocí nástroje Puppet a znovupoužití konfigurace. V této kapitoly bude také popsán firemně kulturní aspekt všech prováděných změn.

6.1 Porovnání stavu před a po optimalizaci

Následující text popisuje některé skutečné události, ke kterým došlo, a důvody, proč by k nim při aktuálním použití DevOps nástrojů, technik a myšlenek nedošlo (nebo je riziko vzniku případně velikost dopadu výrazně nižší).

6.1.1 Nepřístupné informace z běhu aplikace

Firma spolupracovala jako subdodavatel v projektu pro zahraniční firmu. Vyvíjená aplikace běžela na serverech zákazníka. Při řešení konkrétního problému v aplikaci hlášeného od klienta došlo k situaci, kdy nikdo nebyl schopen získat chybový log z běžící aplikace. Po složitých „politických“ jednáních byl zřízen jeden SSH přístup pro celou firmu přímo na produkční server.

To ale v praxi může znamenat problém s možným únikem citlivých dat. Současně je velmi složité se dostat k potřebným informacím. Kdyby byl použit systém pro centrální logování, mohly by se snáze řešit přístupy a získání samotných logů by bylo pro všechny zúčastněné strany jednodušší.

6.1.2 Špatná komunikace mezi oddělením vývoje a provozu

Se stejnou firmou jako v předchozím bodě byly i další problémy, které se vyskytly i při jiných činnostech, kde byla nutná koordinace provozu a vývoje.

Nasazení nových verzí aplikace byl složitý manuální proces, který dělal pouze jeden člověk (který později firmu opustil), takže ve výsledku bylo nasazování předáno na stranu naší firmy i přes to, že v projektu figurujeme jen jako subdodavatel jedné části aplikace. V druhé firmě nezůstal žádný člověk, které by věděl, jak aplikace funguje v produkčním prostředí.

Při použití nástrojů jako Puppet pro konfiguraci produkčních prostředí a Jenkins pro kontinuální integraci a doručování by byl celý vývoj mnohem rychlejší. Snadněji a častěji by se doručovaly nové verze do produkčního prostředí.

6.1.3 Složitá konfigurace testovacího prostředí a migrace serveru

Vytvoření nového prostředí byl pro většinu aplikací velice složitý a zdlouhavý proces, který nikdo z vývojářů nepodstupoval rád a raději se mu vyhýbal. S DevOps nástroji jako jsou Vagrant a

Puppet je jednoduché předpřipravit konfigurace pro jednotlivé virtuální stroje, které budou představovat konkrétní aplikace.

Pro běžný vývoj není virtuální stroj z důvodu výkonu vhodný, nicméně pro testovací účely je použití výše zmíněných nástrojů ideální. V případě potřeby vytvoření dalšího prostředí na fyzických serverech to s nástrojem Puppet bude znamenat jen několik řádků konfigurace a vše se automaticky nastaví.

Migrace serveru od jednoho poskytovatele k druhému, ke které došlo, byla bez nástroje Puppet velmi zdlouhavá a riziková. S použitím nástroje Puppet by podobná migrace v tuto chvíli byla otázkou několika minut.

6.1.4 Nekonzistentní sestavení

Aplikace pohodlne.info, skládající se s více modulů, byla v první fázi (viz kapitola 5.4.1) sestavována a nasazována po jednotlivých modulech. Při jedné změně rozhraní modulu ale vznikla nekompatibilita s modulem, který danou funkčnost využíval.

Chyba se objevila po velmi dlouhém a složitém hledání až na testovacím serveru. Hlavní problém byl v tom, že přestože byly všechny moduly v jednom repozitáři, bylo možné jejich sestavení a nasazení do testovacího prostředí spouštět samostatně. Při změně rozhraní se sestavoval jen změněný modul, který se nasadil. Modul, který jeho funkce využíval, přestal fungovat. Vývojář nefunkční modul neodhalil, protože s ním nepracoval a neměl ho otevřen v IDE.

Díky tomu, že se všechny moduly sestavují a nasazují společně, se tato situace už nemůže opakovat. Případná nekompatibilita na úrovni kompilace by se objevila už při sestavení na Jenkins serveru a vůbec by nedošlo k nasazení porušené aplikace.

6.2 Počet sestavení a doba nasazení změn

Původní fungování nasazování do testovacího i produkčního prostředí bylo založeno na principu žádosti. Žádost o sestavení podával buď vývojář, který vyvinul novou funkci a potřeboval funkčnost vyzkoušet v dalším prostředí, nebo tester, který měl za úkol nové funkčnosti na testovacím serveru vyzkoušet.

Určený vývojář potom spustil příslušné skripty, sestavené artefakty nahrál manuálně na požadované místo a provedl nasazení. Tím, že šlo převážně o manuální práci, nedocházelo k nasazení nových verzí tak často, jak by mělo. S použitím DevOps nástrojů a technik se počet nasazení do testovacího prostředí výrazně zvýšil[20]. Aplikace pohodlne.info se automaticky sestavuje a nasazuje minimálně 1x denně v nočních hodinách, kdy nevadí větší vytížení serveru a chvilková nedostupnost testovacího prostředí. Aplikace EditorŠVP, která je mnohem menší a její sestavení probíhá rychleji, se sestavuje a nasazuje 2x denně. Sestavení neproběhne vůbec, pokud neexis-

[20]Cílem je zvýšit počet nasazení do produkčního prostředí, nicméně zvýšení počtu v testovacím je první krok.

tují změny ve zdrojových souborech. Ke kontinuálnímu nasazování chybí jen změna ze spouštění úloh na základě času na spouštění na základě změny ve zdrojových kódech.

6.3 Rozdělení testovacího aplikačního serveru

Tato podkapitola se zaměřuje na důvody, které vedly k rozdělení testovacího serveru, a na řešení prostřednictvím nástroje Puppet.

Původně existoval na testování jen jeden aplikační server Tomcat, na kterém běželo velké množství aplikací. Aplikace, které na tomto serveru běžely, se v produkčním prostředí nepotkávaly a v některých případech vyžadovaly rozdílné nastavení. Testovací server tak byl hodně rozdílný oproti serveru produkčnímu.

Jeden Tomcat sice znamenal méně práce (resp. po instalaci žádnou) pro správce daného serveru, ale ze strany vývojářů a testerů to představovalo problém. Restart celého Tomcatu (v případě problému nebo nové verze aplikace) s více než 25 aplikacemi trval velmi dlouho a bylo nutné kvůli tomu zastavit práci lidí na všech aplikacích, i když se prováděla aktualizace jen jedné z nich.

Rozdělení aplikací na více samostatných serverů by ale bez DevOps postupů a nástrojů bylo složitější. Instalace více aplikačních serverů znamená více úkolů pro správce i větší nároky na vývojáře. Správce musí vědět, na který server daná aplikace (nebo modul) patří. Musí hlídat funkčnost více serverů, při změnách konfigurací provést změny na dvou místech atd. Tyto problémy elegantně řeší Puppet.

Rozdělení může představovat i pro vývojáře zvýšení pozornosti a nároků. Vývojáři mají na tomto serveru přístup k surovým logům, takže v případě dvou aplikačních serverů musí znát obě cesty a vědět, ve které hledat informace, které zrovna potřebují. Stejná situace nastává i při provádění vlastních nasazení. Pokud se jedná o logy pro centrální logování, s Graylogem půjde jen o další zdroj dat a vývojáři k nim budou přistupovat stejně jako by šlo o data z jednoho serveru.

Dále jsem také učinil rozhodnutí, že by testeři i vývojáři nemuseli přesně vědět, na kterém hardwaru jejich webová aplikace pro testovací účely běží. V původní verzi se na společný Tomcat přistupovalo přes adresu odkazující na název serveru. Převodem na adresy aplikací, které jsou blízké produkčním, jsem získal možnost aplikační servery přesouvat mezi hardwarem bez vědomí uživatelů. Současně jsem firmu použitím proxy serveru zbavil nutnosti používat porty pro přístup k jednotlivým aplikacím[21].

Dva nové aplikační servery vznikly kopií databáze, dat a aplikací z původního serveru. Pouhým odstraněním některých aplikací v jednom aplikačním serveru se viditelně zvýšila rychlost startu. Toto zrychlení umožní mnohem flexibilnější nasazování nových verzích i rychlejší testování, což je jeden z cílů DevOps. Dále se po tomto rozdělení obě instance přiblížily více pro-

[21] Zde platí DevOps pravidlo o zjednodušení přístupu pro všechny, zadávání čísla portů je pro méně technicky zdatné testery nebo testující zákazníky problém.

dukčnímu prostředí a v případě potřeby bude možné do nich snadno obnovit produkční data. Zrychlení aplikace i možnost častějšího nasazování nových verzí působí na vývojáře i testery dobře a jejich práce je produktivnější. Taky se tím otevřela cesta pro automatické nasazování změn, protože je možné změny nasazovat stejně jako do produkčního prostředí.

Nevýhodou rozdělení je vyšší zatížení zdrojů fyzického serveru.

Díky použití nástroje Puppet se s dalším aplikačním serverem nezvýšila složitost správy. Konfigurace je na jednom místě a znovupoužitelná. Použitím centrálního logování a nasazováním prostřednictvím nástroje Jenkins se nezvýšily nároky ani na vývojáře.

6.4 Testování konfigurace před změnou

Díky použití nástroje Puppet lze každou změnu konfigurace serverů vyzkoušet v testovacím virtuálním prostředí bez ohrožení funkčnosti reálného serveru. Pomocí jednoho příkazu lze v poměrně krátkém čase získat prostředí shodné s produkčním a vyzkoušet změny, které se potom propagují do produkčního prostředí.

Díky možnosti spouštět Puppet s parametrem `noop`, lze i před provedením konfigurace na fyzickém serveru simulovat provádění změn. Puppet pouze oznámí, ke kterým změnám by došlo při reálném běhu.

Při práci na rozdělení Tomcatu jsem pro Puppet konfiguraci vytvořil větev v systému Git a úpravy dělal odděleně – snahou bylo získat výsledek co nejrychleji. Pro následné spojení změn jsem využil možnost získat konfigurační kód, který je aktuálně na produkčním prostředí, a vyzkoušet jakým způsobem zareaguje na změny, které jsem při úpravách udělal.

6.5 Změny v myšlení

Kromě nástrojů je v DevOps kladen také velký důraz na nastavení myšlení všech účastníků vývoje a provozu daného software. Tato část je důležitá kvůli faktu, že ať bude nástroj sebelepší, tak pokud lidmi nebude přijat, nikdy nebude fungovat správně.

Vývojáři se naučili odevzdávat změny do Git repozitáře zodpovědněji (kompletní a otestované), protože ví, že se změna hned objeví a tester nebo management ji bude moct zkontrolovat.

Dříve se stávalo, že se před schůzkou, na které probíhá hodnocení předchozích úkolů, na poslední chvíli provádělo nasazení, aby bylo na schůzce po několika denní práci co ukázat. S automatickým nočním nasazováním se tahle situace už neopakuje.

Stejně tak docházelo k situacím, kdy tester nevěděl, která funkčnost už je připravena k testování. V systému Redmine byl daný úkol označený jako splněný, ale z běžícího systému nebylo jasné, zda je tam změna skutečně nasazena. Díky automatickému nasazování již nemůže tato situace nastat. Navíc díky automatické změně stavu při nasazení je v systému Redmine mnohem větší přehled.

Dalším zajímavým faktem je, že se zvýšila délka zpráv a počet jednotlivých commitů (počítáno v repozitáři aplikace pohodlne.info). V dobách před záváděním DevOps praktik někteří vývojáři svou práci označovali jen slovy „změny“. V současné době se snaží o lepší popisy. Pokud se jedná přímo o úkol ze systému Redmine, připojují i jejich čísla – sami pochopili význam těchto zpráv.

Vzhledem k nově otevřeným možnostem, jako například pravidelné spouštění testů a zavedení statického testování, se začalo o těchto možnostech mnohem víc mluvit i ze strany managementu.

Vývojáři stále přicházejí s nápady na vylepšení celého procesu, ať už čistě v oblasti vývoje nebo v oblasti provozu. Například tester si aktivně vyžádal přístup do aplikace Graylog, když se testovaná aplikace začala chovat nestandardně. Vývojáři tak bylo vytvořeno hlášení už s kompletním popisem chyby včetně chybového logu z testovací serveru.

Celá kapitola se zabývala analýzou dopadu provedených změn na vývojový proces. V první části bylo popsáno několik skutečných problému, které by v tuto chvíli s technikami DevOps nenastaly. Další část se zaměřuje na rozdíly vzniklé po nasazení technik DevOps a popisuje problémy, které se řešily až s použitím DevOps přístupu. Poslední část popisuje přijetí DevOps myšlenek napříč celou firmou.

7 Závěr

V této práci jsem popsal momentálně populární přístup DevOps, jeho historii, myšlenky a konkrétní využívané praktiky. Práce je z poloviny zaměřená na konkrétní firmu, její produkty a způsob vývoje, ale informace v ní obsažené je možné použít obecně v jakékoliv firmě zabývající se vývojem software.

V průběhu řešení jsem porozuměl velkému množství praktik, které jsem teoreticky popsal a v praxi aplikoval. Nasadil jsem velké množství nástrojů, se kterými jsem před touto prací neměl žádné zkušenosti, a posunul tak vývoj a provoz ve firmě VRK plus s.r.o. na vyšší úroveň.

Technologie a nástroje, se kterými jsem pracoval, jsou aktuální a mají široké uplatnění a některé by svou složitostí a rozmanitostí vystačily na samostatnou diplomovou práci. Některé nástroje nebo techniky jsou, na úkor podrobnostem, popsány stručně a bez technických detailů, ale je třeba připomenout, že tento text má sloužit jako odrazový můstek do dané problematiky. Tuto úlohu text plní a po jeho přečtení by měl člověk pracující v softwarové firmě chápat základní myšlenky DevOps a být schopen je uplatnit v praxi.

V posledních kapitolách rozebírám několik reálných situací, které by byly nyní, po nasazení některých DevOps praktik a nástrojů, snadněji řešitelné a analyzuji celkový dopad na vývojový proces firmy. Některé situace, které znamenaly zdržení ve vývoji software, by v současnosti vůbec nenastaly.

V započaté práci dále aktivně pokračuji a podílím se na rozvoji používaných nástrojů, nastavených procesů a bourání bariér mezi vývojem a provozem. Během tvorby této diplomové práce jsem dělal v rámci tří předmětů navazujícího studia prezentace, abych kolegům celou oblast přiblížil a zjistil jejich názory na tuto problematiku. Současně jsem také různými formami předával získané poznatky a vědomosti ve firmě VRK plus s.r.o.

Literatura

[1] MUELLER, Ernest. What Is DevOps. *The agile admin* [online]. 2010, 7.12.2016 [cit. 2017-04-02]. Dostupné z: https://theagileadmin.com/what-is-devops/

[2] KIM, Gene. The Amazing DevOps Transformation Of The HP LaserJet Firmware Team (Gary Gruver). *IT Revolution* [online]. Portland, 2014 [cit. 2017-04-10]. Dostupné z: http://itrevolution.com/the-amazing-devops-transformation-of-the-hp-laserjet-firmware-team-gary-gruver/

[3] DEBOIS, Patrick. Agile Infrastructure & Operations. In: *Agile 2008 Toronto* [prezentace]. [cit. 2017-01-15]. Dostupné z: http://www.jedi.be/blog/2008/10/09/agile-2008-toronto-agile-infrastructure-and-operations-presentation/

[4] SHARMA, Sanjeev. *DevOps For Dummies*. United States of America: John Wiley & Sons, 2014. ISBN 978-1-118-73378-3.

[5] *DevOps Topologies* [online]. 2016 [cit. 2017-04-05]. Dostupné z: http://web.devopstopologies.com/

[6] There's No Such Thing as a "Devops Team". *Continuous Delivery* [online]. 2012 [cit. 2017-04-05]. Dostupné z: https://continuousdelivery.com/2012/10/theres-no-such-thing-as-a-devops-team/

[7] ALLSPAW, John a Paul HAMMOND. 10+ Deploys Per Day: Dev and Ops Cooperation at Flickr. In: *O'Reilly.com Velocity 2009* [prezentace]. [cit. 2017-01-15]. Dostupné z: http://conferences.oreilly.com/velocity/velocity2009/public/schedule/detail/7641

[8] *DevOpsDays: The conference that brings development and operations together.* [online]. 2017 [cit. 2017-01-15]. Dostupné z: https://www.devopsdays.org/events/

[9] 2013 State of DevOps Report. *Puppet* [online]. IT Revolution Press, 2013 [cit. 2017-04-05]. Dostupné z: https://puppet.com/resources/whitepaper/2013-state-devops-report

[10] DevOps in 2016: The Year of Implementation. *DevOps.com* [online]. 2015 [cit. 2017-04-10]. Dostupné z: https://devops.com/devops-2016-year-implementation/

[11] BEYER, Betsy, Chris JONES, Jennifer PETOFF a Niall Richard MURPHY, ed. *Site Reliability Engineering: How Google Runs Production Systems*. 1. O'Reilly Media, 2016. ISBN 978-1491929124.

[12] ET AL. BECK, K. Manifesto for Agile Software Development [online]. 2001 [cit. 2017-03-03]. Dostupné z: http://agilemanifesto.org/

[13] HüTTERMANN, Michael. *DevOps for developers*. New York: Distributed to the book trade worldwide by Springer Science Business Media New York, c2012, xviii, 176 p. Expert's voice in Web development. ISBN 14-302-4569-7.

[14] BOOCH, Grady. *Object oriented design with applications*. Redwood City, Calif.: Benjamin/Cummings Pub. Co., 1991. ISBN 978-080-5300-918.

[15] Continuous Integration. *Martin Fowler* [online]. 2006 [cit. 2017-04-03]. Dostupné z: `https://martinfowler.com/articles/continuousIntegration.html`

[16] ContinuousDelivery. *Martin Fowler* [online]. 2013 [cit. 2017-04-03]. Dostupné z: `https://martinfowler.com/bliki/ContinuousDelivery.html`

[17] What is DevOps? *Ecole informatique - SUPINFO International University* [online]. Mauricius, 2016 [cit. 2017-03-14]. Dostupné z: `https://www.supinfo.com/articles/single/3652-what-is-devops`

[18] LOELIGER, Jon. *Version control with Git*. Sebastopol, CA: O'Reilly, c2009. ISBN 978-059-6520-120.

[19] SINK, Eric. *Version control by example*. Champaign, Ill.: Pyrenean Gold Press. ISBN 978-098-3507-901.

[20] BASS, Len., Ingo M. WEBER a Liming ZHU. *DevOps: a software architect's perspective*. ISBN 978-013-4049-847.

[21] AIELLO, Bob a Leslie A. SACHS. *Configuration management best practices: practical methods that work in the real world*. Upper Saddle River, NJ: Addison-Wesley, c2011. ISBN 978-032-1685-865.

[22] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu*. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6.

[23] VRK PLUS S.R.O. VRK plus s.r.o.: Zakázkový vývoj software a webových aplikací [online]. 2014 [cit. 2017-02-28]. Dostupné z: `http://www.vrk.cz/`

[24] VERONA, Joakim. *Practical DevOps*. Packt Publishing, 2016. ISBN 9781785882876.

[25] Monorepo. *Borekb.cz* [online]. 2017 [cit. 2017-03-26]. Dostupné z: `https://borekb.cz/2017/01/monorepo/`

[26] Advantages of monolithic version control. *Dan Luu Blog* [online]. 2015 [cit. 2017-03-26]. Dostupné z: `http://danluu.com/monorepo/`

[27] FeatureBranch. *Martin Fowler* [online]. 2009 [cit. 2017-03-27]. Dostupné z: `https://martinfowler.com/bliki/FeatureBranch.html`

[28] GAJDA, Włodzimierz. *Pro Vagrant*. Apress, 2015. ISBN 978-148-4200-742.

[29] TURNBULL, James. a Jeffrey. MCCUNE. *Pro Puppet*. S.l.: Apress, 2010. ISBN 14-302-3057-6.

[30] KHARE, Tanuj. Apache Tomcat 7 essentials. Mumbai: Packt Pub., 2012. Community experience distilled. ISBN 978-1-84951-663-1.

A Soubory na přiloženém CD

Příloha obsahuje ukázkovou konfiguraci používaných nástrojů. Pomocí nástroje Vagrant se připraví dva virtuální stroje, které simulují produkční prostředí (virtuální stroj `pohodlne.info`) a prostředí pro testování a provoz podpůrných nástrojů (`vyvoj.vrk.cz`). Ukázka rámcově odpovídá konfiguraci prováděné během této diplomové práce ve firmě VRK plus s.r.o. Obsahuje strukturu i testovací data.

Konfigurace na serveru `vyvoj.vrk.cz` obsahuje nástroje spuštěné v Docker kontejnerech (Redmine, Jenkins, GitLab a Graylog). Na server `pohodlne.info` se nainstaluje aplikace Liferay, logy z této aplikace proudí do systému GrayLog. Do systému GitLab lze naimportovat předpřipravený projekt, který se automaticky sestaví a nasadí v systému Jenkins, odkaz na úkol v systému Redmine způsobí změnu stavu. Aplikace se nasadí do prostředí `pohodlne.info`.

Obsah CD

puppet – složka se strukturou pro program Puppet, který se stará o konfiguraci.

vms/Vagrantfile – soubor s definicí testovacích virtuální strojů, které se po spuštění nakonfigurují podle Puppet skriptu.

example-project – adresář s Git repozitářem obsahující ukázkový Liferay projekt.

import – adresář s ukázkovými daty.

README.txt – soubor s instrukcemi a popisem konfigurace.

B Skripty pro změnu stavu úkolu v systému Redmine při sestavení v nástroji Jenkins

```
1  import hudson.model.*;
2  import hudson.util.*;
3  import hudson.scm.*;
4  import hudson.plugins.accurev.*
5  import java.util.regex.*;
6
7  def thr = Thread.currentThread();
8  def build = thr?.executable;
9  def changeSet= build.getChangeSet();
10
11 List<hudson.plugins.git.GitChangeSet> items = changeSet.getItems();
12
13 def issuesNumber = [];
14 for(item in items){
15    Pattern pattern = Pattern.compile("(#)\\w+");
16    Matcher matcher = pattern.matcher(item.getComment());
17
18    if(matcher.find()){
19       issuesNumber.add(matcher.group(0).substring(1));
20       while(matcher.find()){
21          issuesNumber.add(matcher.group(0).substring(1));
22       }
23    }
24 }
25
26 def addIssuesNumber = new ParametersAction([
27       new StringParameterValue("ISSUE_NUMBERS",
28          issuesNumber.toString().replace("[", "").replace("]","").replace(","," "
               ))
29     ])
30
31 build.addAction(addIssuesNumber)
```

Výpis 4: Skript, který je součástí úlohy pro sestavení a nasazení. Skript v jazyce Groovy prochází jednotlivé změny a hledá pattern odpovídající číslu úkolu v systému Redmine.

```
#!/bin/bash

#ISSUE_NUMBERS="7819 7636 7635 8612"
#STATUS_ID="9"
#NOTE="Zmeneno #28 test5"
NOTE=$3
NOTE=${NOTE// /_}

for NUMBER in $ISSUE_NUMBERS
do
    echo $NUMBER
    echo '{"issue": {"status_id": "'$STATUS_ID'", "notes": "'$NOTE'" } }'

    curl -v -H "Content-Type: application/json" \
           -H "X-Redmine-API-Key: xxxx" \
           -X PUT -d '{"issue": {"status_id": "'$STATUS_ID'", "notes": "'$NOTE'
               " } }' http://helpdesk.vrk.cz/issues/$NUMBER.json
done
```

Výpis 5: Skript v jazyce Bash, který provádí prostřednictvím volání API samotnou změnu stavu v systému Redmine.

C Vagrantfile pro testování konfigurace

```
nodes = [
  { :hostname => 'pohodlne.info', :ip => '192.168.56.42', :box => 'cargomedia/
      debian-8-amd64-default', :ram => 2048,
    :ports => [
      {
        :guest => '80',
        :host => '8088'
      },
      {
        :guest => '443',
        :host => '8043'
      },
      {
        :guest => '8000',
        :host => '8000'
      }
    ]
  },
  { :hostname => 'vyvoj.vrk.cz', :ip => '192.168.56.44', :box => 'cargomedia/
      debian-8-amd64-default', :ram => 3048,
    :ports => [
      {
        :guest => '9000',
        :host => '9009'
      },
      {
        :guest => '12201',
        :host => '12201',
        :protocol => 'udp'
      },
      {
        :guest => '12201',
        :host => '12201',
        :protocol => 'tcp'
      }
      {
        :guest => '8082',
```

```
      :host => '8082'
    }
  ]
 }
]

Vagrant.configure("2") do |config|
  nodes.each do |node|
    config.vm.define node[:hostname] do |nodeconfig|
      nodeconfig.vm.box = node[:box]
      nodeconfig.vm.hostname = node[:hostname]
      nodeconfig.vm.network :private_network, ip: node[:ip]

      if !node[:ports].nil?
        node[:ports].each do |port|
            if !port[:portocol].nil?
              nodeconfig.vm.network :forwarded_port, guest: port[:guest], host:
                  port[:host], protocol: port[:protocol]
            else
              nodeconfig.vm.network :forwarded_port, guest: port[:guest], host:
                  port[:host]
            end
        end
      end

      memory = node[:ram] ? node[:ram] : 256;
      nodeconfig.vm.provider :virtualbox do |vb|
        vb.customize [
          "modifyvm", :id,
          "--cpuexecutioncap", "50",
          "--memory", memory.to_s,
          "--clipboard", "bidirectional",
        ]
      end
    end
  end

   config.vm.provision "shell", :inline => <<-SHELL
    apt-get update
```

```
    apt-get purge puppet
    apt-get install -y puppet-common
    apt-get install -y puppet
  SHELL


  config.vm.provision :puppet do |puppet|
    puppet.manifests_path = "git/puppet/manifests"
    puppet.manifest_file = "site.pp"
    puppet.module_path = "git/puppet/modules"
    puppet.options = "--verbose --debug"
  end
end
```

Výpis 6: Předpis pro nástroj Vagrant, který popisuje dva virtuální servery pro otestování konfigurace prostředí provedené nástrojem Puppet.

D Zkrácený Puppet skript

```
node 'pohodlne.info' {
  $pcg = ['screen', 'maven', 'curl', 'mc', 'zip']
  package { $pcg: ensure => 'installed' }

  class { 'oracle_java':
    version       => '8u45',
    add_system_env => true,
    add_alternative => true
  }

  class { 'mysql::server':
    root_password         => '***',
    remove_default_accounts => true,
    override_options      => $override_options,
  }

  class { 'tomcat':
    user    => 'tomcat7',
    require => Class['oracle_java']
  } ->
  tomcat::instance { 'tomcat7':
    source_url   => 'https://archive.apache.org/dist/tomcat/tomcat-7/v7.0.69/
        bin/apache-tomcat-7.0.69.tar.gz',
    catalina_base => '/opt/tomcat-7/',
  } ->
  tomcat::service { 'tomcat7':
    service_ensure => running,
    catalina_base => '/opt/tomcat-7',
  } ->
  tomcat::configServer { 'tomcat-7':
    catalina_base => '/opt/tomcat-7/',
    shutdown_port => '8005',
    http_port    => '8080',
  }

  tomcat::setenv::entry { 'CATALINA_OPTS':
    param      => 'CATALINA_OPTS',
```

```
37      config_file => '/opt/tomcat-7/bin/setenv.sh',
38      quote_char => '"',
39      value      => [
40        "-Dcom.sun.management.jmxremote.rmi.port=9992",
41        "-Dcom.sun.management.jmxremote.port=9991",
42        "-Djava.rmi.server.hostname=localhost",
43        "-Xms2048m", "-Xmx2048m"],
44    }
45
46    file { '/opt/tomcat-7/conf/tomcat-users.xml':
47      require => Tomcat::Instance['tomcat7'],
48      content => template('template/tomcat-users.xml.erb'),
49      ensure => present,
50      notify => Tomcat::Service['tomcat7']
51    }
52
53    class { 'nginx':}
54
55    file { '/etc/nginx/conf':
56      require => File['/etc/nginx'],
57      ensure => 'directory',
58      owner  => nginx,
59      group  => nginx,
60      mode   => 755,
61    } ->
62    file { '/etc/nginx/conf/uni_pohodlne.info.crt':
63      content => template('template/uni_pohodlne.info.crt'),
64      ensure => present,
65      mode   => 600
66    } ->
67    file { '/etc/nginx/conf/pohodlne.info.key':
68      content => template('template/pohodlne.info.key'),
69      ensure => present,
70      mode   => 600
71    } ->
72    exec { 'create_key':
73      command => 'openssl dhparam -out /etc/nginx/conf/dhparams.pem 2048',
74      path   => '/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin',
75    } ->
```

```
76
77    nginx::resource::vhost { 'pohodlne.info':
78      ssl                 => true,
79      ssl_cert            => '/etc/nginx/conf/uni_pohodlne.info.crt',
80      ssl_key             => '/etc/nginx/conf/pohodlne.info.key',
81      ssl_dhparam         => '/etc/nginx/conf/dhparams.pem',
82      listen_port         => 80,
83      rewrite_to_https    => true,
84      rewrite_www_to_non_www => true,
85      proxy               => 'http://127.0.0.1:8080/',
86    }
87
88    mysql::db { 'lr_625':
89      user     => 'lportal',
90      password => '***',
91      host     => 'localhost',
92    }
93
94    class { 'liferay':
95      tomcat_instalation_dir => '/opt/tomcat-7',
96      lr_version          => '6.2.5',
97      tomcat_instance     => 'tomcat7',
98      require             => Tomcat::Instance['tomcat7'],
99      db                  => {
100       driver   => false,
101       user     => 'lportal',
102       password => '***',
103       db_name => 'lr_625'
104     },
105     liferay_dir         => '/opt/liferay/',
106   }
107 }
```

Výpis 7: Zkráceny a upravený Puppet skript pro instalaci prostředí pro běh aplikace pohodlne.info. Skript obsahuje instalaci běhové prostředí jazyka Java, instalaci MySQL databáze, vytvoření schématu, instalace a konfigurace aplikačního serveru Tomcat.

E Soubor pro kompletní instalaci aplikace GrayLog

```
version: '2'
services:
  mongo:
    image: "mongo:3"
    volumes:
      - /opt/graylog/data/mongo:/data/db
  elasticsearch:
    image: "elasticsearch:2"
    command: "elasticsearch -Des.cluster.name='graylog'"
    volumes:
      - /opt/graylog/data/elasticsearch:/usr/share/elasticsearch/data
  graylog:
    image: graylog2/server:2.1.2-1
    volumes:
      - /opt/graylog/data/graylog/journal:/usr/share/graylog/data/journal
      - /opt/graylog/config/graylog:/usr/share/graylog/data/config
      - /opt/graylog/tmp:/tmp/geoloc
    environment:
      GRAYLOG_PASSWORD_SECRET: xxx
      GRAYLOG_ROOT_PASSWORD_SHA2: xxx
      GRAYLOG_WEB_ENDPOINT_URI: http://192.168.2.2:9000/api/
    depends_on:
      - mongo
      - elasticsearch
    ports:
      - "9000:9000"
      - "19000:19000"
      - "12201/udp:12201/udp"
      - "9514/udp:9514/udp"
      - "12201/tcp:12201/tcp"
      - "12202/udp:12202/udp"
      - "1514/udp:1514/udp"
```

Výpis 8: Soubor `docker-compose.yaml` pro konfiguraci tří kontejnerů, které jsou pro provoz Graylogu potřebné.